デベロッパーガイド

AWS Elastic Beanstalk

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Elastic Beanstalk: デベロッパーガイド

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスはAmazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

とは AWS Elastic Beanstalk 1
サポートされているプラットフォーム 1
アプリケーションのデプロイワークフロー 1
料金 2
次のステップ 2
入門チュートリアル 3
構築する内容 3
ステップ 1 - アプリケーションを作成する 4
ステップ 2 - アプリケーションをデプロイする 6
ステップ 3 - 環境を調べる 8
ログによるトラブルシューティング 9
ステップ 4 - アプリケーションを更新する 11
ステップ 5 - アプリケーションをスケールする 12
容量設定を増やす 12
容量の増加を検証する 13
クリーンアップ 13
次のステップ 15
EB CLI のセットアップ 16
EB CLI のインストール 17
手動インストール 18
virtualenv に をインストールする 20
homebrew で EB CLI をインストールする 21
EB CLI の設定 22
.ebignore を使用してファイルを無視する 24
名前を指定されたプロファイルを使用する 25
プロジェクトフォルダの代わりにアーティファクトをデプロイする 26
構成設定と優先順位 26
インスタンスメタデータ 27
Git での EB CLI の使用 27
Elastic Beanstalk 環境を Git ブランチに関連付ける 28
変更のデプロイ 28
Git サブモジュールの使用 29
Git タグのアプリケーションバージョンへの割り当て 30
EB CLI コマンド 31

一般的なオプション 32
eb abort 33
説明 33
構文 33
オプション 33
Output 33
例 33
eb appversion 34
説明 34
構文 34
オプション 34
コマンドのインタラクティブな使用 36
Output 37
例 37
eb clone 38
説明 38
構文 38
オプション 39
Output 40
例 40
eb codesource 41
説明 41
構文 41
オプション 41
Output 41
例 42
eb config 42
説明 42
構文 43
オプション 44
Output 47
例 48
eb console 51
説明 51
構文 52
オプション 52

eb create 52
説明 52
構文 53
オプション 53
Output 66
例 67
eb deploy 69
説明 69
構文 69
オプション 70
Output 71
例 71
eb events 71
説明 71
構文 71
オプション 72
Output 72
例 72
eb health 73
説明 73
構文 73
オプション 73
Output 74
例 74
eb init 75
説明 75
構文 75
オプション 76
CodeBuild サポート 78
Output 78
例 78
eb labs 80
説明 80
eb list 80
説明 80
構文 80

オプション 80
Output 81
例 1 81
例 2 81
eb local 81
説明 81
構文 82
オプション 82
Output 83
例 84
eb logs 85
説明 85
構文 85
オプション 86
Output 89
例 89
eb migrate 89
説明 89
構文 91
サブコマンド 91
オプション 93
Output 101
例 101
eb open 105
説明 105
構文 106
オプション 106
Output 106
eb platform 106
説明 106
カスタム プラットフォーム用に eb プラットフォームの使用 107
環境に eb プラットフォームを使用 114
eb printenv 117
説明 117
構文 117
オプション 117

Output 117
例 117
eb restore 117
説明 117
構文 118
オプション 118
Output 118
例 118
eb scale 119
説明 119
構文 119
オプション 119
Output 119
例 119
eb setenv 120
説明 120
構文 120
オプション 120
Output 120
例 120
eb ssh 121
説明 121
構文 122
オプション 122
Output 123
例 123
eb status 124
説明 124
構文 124
オプション 124
Output 125
例 125
eb swap 126
説明 126
構文 126
オプション 126

Output 127
例 127
eb tags 127
説明 127
構文 128
オプション 128
Output 130
例 130
eb terminate 131
説明 131
構文 132
オプション 132
Output 132
例 133
eb upgrade 133
説明 133
構文 134
オプション 134
Output 134
例 134
eb use 135
説明 135
構文 135
オプション 135
概念 136
アプリケーション 136
アプリケーションバージョン 136
環境 136
環境枠 136
環境設定 137
保存された設定 137
プラットフォーム 137
「ウェブサーバー環境」 137
ワーカー環境 139
設計上の考慮事項 139
スケーラビリティ 140

セキュリティ ... 140
永続ストレージ ... 141
耐障害性 ... 142
コンテンツ配信 ... 143
ソフトウェアの更新プログラムと修正プログラム ... 143
接続 ... 143
アプリケーションの管理 ... 145
アプリケーション管理コンソール ... 146
アプリケーションバージョンの管理 ... 147
アプリケーションバージョンの作成 ... 148
アプリケーションバージョンの削除 ... 148
バージョンライフサイクル ... 150
アプリケーションバージョンのタグ付け ... 152
ソースバンドルを作成する ... 154
コマンドラインからソースバンドルを作成する ... 155
Git を使用してソースバンドルを作成する ... 156
Mac OS X Finder または Windows エクスプローラでファイルを圧縮する ... 156
.NET アプリケーションのソースバンドルの作成 ... 157
ソースバンドルをテストする ... 158
CodeBuild を使用した構築 ... 158
Creating an application ... 159
アプリケーションコードのビルドとデプロイ ... 159
アプリケーションのタグ付け ... 161
アプリケーションの作成時にタグを追加する ... 162
既存のアプリケーションのタグを管理する ... 162
リソースのタグ付け ... 163
タグを付けることができるリソース ... 165
起動テンプレートへのタグの伝播 ... 165
環境の作成 ... 167
環境マネジメントコンソール ... 168
コンソールにアクセスする ... 169
環境の概要のペイン ... 169
環境の詳細 ... 171
環境アクション ... 173
環境の作成 ... 175
新しい環境の作成ウィザード ... 180

環境のクローンを作成する 189
環境を終了する 191
で AWS CLI 193
API の使用 195
Launch Now URL 199
環境を構成する 203
環境を構成する 206
デプロイ 208
デプロイポリシーの選択 210
新しいアプリケーションバージョンのデプロイ 213
以前のバージョンの再デプロイ 213
アプリケーションをデプロイするその他の方法 213
デプロイオプション 214
Blue/Green デプロイ 221
CI/CD の統合 222
設定変更 223
ローリング更新 225
イミュータブルな更新 230
プラットフォームの更新 233
方法 1 – 環境のプラットフォームバージョンを更新する 236
方法 2 – Blue/Green デプロイを実行する 237
マネージド更新 238
レガシー環境をアップグレードする 245
AL2023/AL2 への移行 246
プラットフォームの廃止に関するよくある質問 264
更新のキャンセル 269
環境の再構築 270
実行中の環境の再構築 270
終了した環境の再構築 271
環境タイプ 272
負荷分散されたスケーラブルな環境 272
シングルインスタンス環境 273
環境タイプの変更 273
ワーカー環境 274
ワーカー環境 SQS デーモン 275
デッドレターキュー 277

定期的なタスク 278
ワーカー環境枠での自動スケーリングのための Amazon CloudWatch の使用 279
ワーカー環境の設定 280
環境リンク 282
無効なスタック状態からの復旧 284
エラーへの対応 284
エラーが発生する理由 284
環境の設定 286
プロビジョニングされたリソース 286
コンソールを使用した設定 288
設定ページ 288
[変更の確認] ページ 290
Amazon EC2 インスタンス 290
Amazon EC2 インスタンスタイプ 292
コンソールを使用した設定 293
EC2 セキュリティグループの管理 299
AWS CLIを使用した設定 301
名前空間を使用した設定 311
IMDS 312
Auto Scaling グループ 315
起動テンプレート 317
スポットインスタンスのサポート 320
トリガー 336
スケジュールに基づくアクション 338
ヘルスチェックの設定 342
ロードバランサー 344
Classic Load Balancer 345
Application Load Balancer 353
共有 Application Load Balancer 364
Network Load Balancer 375
アクセスログの設定 382
データベース 383
データベースのライフサイクル 384
コンソールを使用して環境に Amazon RDS DB インスタンスを追加する 384
データベースに接続 387
コンソールを使用した統合 RDS DB インスタンスの設定 387

設定ファイルを使用した統合 RDS DB インスタンスの設定 388
コンソールを使用した RDS DB インスタンスのデカップリング 389
設定ファイルを使用して RDS DB インスタンスをデカップリングする 391
セキュリティ 393
環境セキュリティの設定 393
環境セキュリティ設定の名前空間 395
環境のタグ付け 396
環境の作成時にタグを追加する 396
既存環境のタグの管理 397
環境変数とソフトウェア設定 399
プラットフォーム固有の設定を構成する 399
環境プロパティ (環境変数) の設定 400
ソフトウェア設定の名前空間 402
環境プロパティへのアクセス 404
デバッグ 405
ログの表示 408
通知 410
Elastic Beanstalk コンソールを使用した通知の設定 411
設定オプションを使用した通知の設定 412
通知を送信するためのアクセス許可の設定 414
Amazon VPC 416
Elastic Beanstalk コンソールでの VPC 設定の定義 416
aws:ec2:vpc 名前空間 418
EC2-Classic から VPC への移行 419
ドメイン名 423
環境の設定 (アドバンスト) 425
設定オプション 426
優先順位 426
推奨値 427
環境を作成する前 429
作成時 435
作成後 441
汎用オプション 451
プラットフォーム固有のオプション 539
カスタム オプション 552
.Ebextensions 553

オプション設定 ... 555
[Linux サーバー] ... 557
Windows Server ... 574
カスタムリソース ... 584
保存された設定 ... 612
保存された設定にタグ付けする ... 616
`env.yaml` ... 619
カスタムイメージ ... 622
カスタム AMI の作成 ... 622
カスタム AMI の管理 ... 626
カスタム AMI をクリーンアップする ... 627
廃止されたプラットフォームに基づく AMI ... 627
静的ファイル ... 634
コンソールを使用した静的ファイルの設定 ... 634
構成オプションを使用した静的ファイルの構成 ... 635
HTTPS ... 636
サーバー証明書 ... 638
ロードバランサーで HTTPS を終端する ... 645
インスタンスで HTTPS を終端する ... 648
エンドツーエンドの暗号化 ... 684
TCP パススルー ... 688
HTTP から HTTPS へのリダイレクト ... 690
[Platforms] (プラットフォーム) ... 692
プラットフォームの用語集 ... 692
責任共有モデル ... 696
プラットフォームのサポートポリシー ... 697
廃止されたプラットフォームブランチ ... 697
90 日間の猶予期間経過後 ... 698
プラットフォームスケジュール ... 699
リソースを計画する ... 699
今後のプラットフォームブランチリリース ... 700
廃止されるプラットフォームブランチのスケジュール ... 700
リタイアしたプラットフォームブランチの履歴 ... 702
サーバーと OS の履歴 ... 707
サポートされているプラットフォーム ... 709
サポートされているプラットフォームとコンポーネント履歴 ... 710

Linux プラットフォーム 710
サポートされている Amazon Linux のバージョン 711
Elastic Beanstalk Linux プラットフォームのリスト 712
インスタンスデプロイワークフロー 713
AL 2 以降で動作する ECS のインスタンスデプロイのワークフロー 714
プラットフォームスクリプトツール 716
Linux プラットフォームの拡張 725
ビルドファイルと Procfile 725
プラットフォームフック 727
設定ファイル 730
リバースプロキシの設定 730
拡張機能を使用したアプリケーションの例 734
.NET のデプロイ (Windows) 736
.NET Core on Windows の QuickStart 737
AWS アカウント 738
前提条件 740
ステップ 1: .NET Core on Windows アプリケーションを作成する 741
ステップ 2: アプリケーションをローカルに実行する 741
ステップ 3: EB CLI を使用して .NET Core on Windows アプリケーションをデプロイする . 742
ステップ 4: Elastic Beanstalk でアプリケーションを実行する 743
ステップ 5 : クリーンアップ 743
AWS アプリケーションの リソース 743
次のステップ 744
コンソールでデプロイする 745
ASP.NET の QuickStart 745
AWS アカウント 746
前提条件 748
ステップ 1: ASP.NET アプリケーションを作成する 748
ステップ 2: アプリケーションをローカルに実行する 750
ステップ 3: を使用して ASP.NET アプリケーションをデプロイする AWS Toolkit for Visual Studio 750
ステップ 4: Elastic Beanstalk でアプリケーションを実行する 750
ステップ 5 : クリーンアップ 751
AWS アプリケーションの リソース 751
次のステップ 752
コンソールでデプロイする 752

開発環境 ... 752
IDE のインストール ... 753
のインストール AWS Toolkit for Visual Studio ... 753
.NET プラットフォーム ... 753
Elastic Beanstalk コンソールでの.NET 環境の設定 ... 754
aws:elasticbeanstalk:container:dotnet:apppool 名前空間 ... 755
メジャーバージョンの移行 ... 756
デプロイマニフェスト ... 759
EC2 Fast Launch ... 784
データベースを追加する ... 786
環境に DB インスタンスを追加 ... 786
ドライバのダウンロード ... 787
データベースへの接続 ... 788
AWS Toolkit for Visual Studio ... 789
ローカルでテストします ... 790
Elastic Beanstalk 環境の作成 ... 790
環境を終了する ... 794
デプロイ ... 795
環境を管理します ... 796
アカウントの管理 ... 807
デバッグ ... 807
監視 ... 808
デプロイツール ... 809
オンプレミスアプリケーションの移行 ... 812
廃止されたコンポーネントに関する推奨事項 ... 812
廃止された Windows 2012 R2 ブランチ ... 812
TLS 1.2 との互換性 ... 813
.NET Core のデプロイ (Linux) ... 815
.NET Core on Linux の QuickStart ... 816
AWS アカウント ... 816
前提条件 ... 818
ステップ 1: .NET Core on Linux アプリケーションを作成する ... 819
ステップ 2: アプリケーションをローカルに実行する ... 819
ステップ 3: EB CLI を使用して .NET Core on Linux アプリケーションをデプロイする ... 820
ステップ 4: Elastic Beanstalk でアプリケーションを実行する ... 822
ステップ 5：クリーンアップ ... 743

AWS アプリケーションの リソース ... 822
次のステップ ... 823
コンソールでデプロイする ... 824
デベロッパー環境 ... 824
.NET Core SDK のインストール ... 824
IDE のインストール ... 753
のインストール AWS Toolkit for Visual Studio ... 753
.NET Core on Linux プラットフォーム ... 825
.NET Core on Linux プラットフォームに関する考慮事項 ... 825
.NET Core on Linux 環境の設定 ... 826
.NET Core on Linux 設定の名前空間 ... 827
アプリケーションのバンドル ... 828
[Procfile] ... 829
プロキシサーバー ... 830
AWS Toolkit for Visual Studio ... 831
前提条件 ... 832
新しいアプリケーションプロジェクトの作成 ... 832
Elastic Beanstalk 環境を作成してアプリケーションをデプロイする ... 833
環境を終了する ... 835
環境を管理します ... 836
監視 ... 847
Windows から Linux への移行 ... 848
.NET Core on Linux プラットフォームへの移行に関する考慮事項 ... 848
Go のデプロイ ... 850
Go の QuickStart ... 850
AWS アカウント ... 851
前提条件 ... 853
ステップ 1: Go アプリケーションを作成する ... 853
ステップ 2: EB CLI を使用して Go アプリケーションをデプロイする ... 854
ステップ 3: Elastic Beanstalk でアプリケーションを実行する ... 855
ステップ 4: クリーンアップする ... 743
AWS アプリケーションの リソース ... 856
次のステップ ... 857
コンソールでデプロイする ... 857
デベロッパー環境 ... 857
Go のインストール ... 857

AWS SDK for Go のインストール ... 858
Go プラットフォーム ... 858
Go 環境の設定 ... 859
Go 設定の名前空間 ... 860
[Procfile] ... 862
[Buildfile] ... 864
プロキシ設定 ... 865
Java のデプロイ ... 867
Java の QuickStart ... 868
AWS アカウント ... 869
前提条件 ... 871
ステップ 1: Java アプリケーションを作成する ... 872
ステップ 2: アプリケーションをローカルに実行する ... 874
ステップ 3: EB CLI を使用して Java アプリケーションをデプロイする ... 875
ステップ 4: Elastic Beanstalk でアプリケーションを実行する ... 876
ステップ 5 : クリーンアップ ... 877
AWS アプリケーションの リソース ... 877
次のステップ ... 878
コンソールでデプロイする ... 878
Tomcat での Java の QuickStart ... 879
AWS アカウント ... 879
前提条件 ... 882
ステップ 1: Java JSP アプリケーションを作成する ... 882
ステップ 2: EB CLI を使用して Java JSP アプリケーションをデプロイする ... 883
ステップ 3: Elastic Beanstalk でアプリケーションを実行する ... 883
ステップ 4: クリーンアップする ... 743
AWS アプリケーションの リソース ... 884
次のステップ ... 885
コンソールでデプロイする ... 885
開発環境 ... 886
Java 開発キットをインストールする ... 886
ウェブ コンテナをインストールする ... 886
ライブラリをダウンロードする ... 886
AWS SDK for Java のインストール ... 887
IDE またはテキストエディタをインストールする ... 887
サンプルアプリケーションとチュートリアル ... 888

サンプル Java アプリケーションで環境を起動する ... 888
次のステップ ... 897
Tomcat プラットフォーム ... 897
Tomcat 環境を設定する ... 899
Tomcat 設定の名前空間 ... 902
WAR ファイルのバンドル ... 904
プロジェクトフォルダーを構築する ... 906
プロキシ設定 ... 909
Java SE プラットフォーム ... 915
Java SE 環境を設定する ... 916
Java SE 設定の名前空間 ... 917
[Buildfile] ... 919
[Procfile] ... 920
プロキシ設定 ... 921
データベースを追加します ... 924
JDBC ドライバーのダウンロード ... 926
データベースへの接続 (Java SE プラットフォーム) ... 927
データベースへの接続 (Tomcat プラットフォーム) ... 928
データベース接続のトラブルシューティング ... 931
リソース ... 933
Node.js のデプロイ ... 934
Node.js の QuickStart ... 934
AWS アカウント ... 935
前提条件 ... 937
ステップ 1: Node.js アプリケーションを作成する ... 938
ステップ 2: アプリケーションをローカルに実行する ... 939
ステップ 3: EB CLI を使用して Node.js アプリケーションをデプロイする ... 939
ステップ 4: Elastic Beanstalk でアプリケーションを実行する ... 940
ステップ 5 : クリーンアップ ... 743
AWS アプリケーションの リソース ... 940
次のステップ ... 941
コンソールでデプロイする ... 942
デベロッパー環境 ... 942
Node.js をインストールします ... 942
npm のインストールを確認します ... 943
AWS SDK for Node.js をインストールする ... 943

Express ジェネレーターをインストールする 943
Express フレームワークとサーバーを設定する 944
Node.js プラットフォーム 945
Node.js 環境の設定 946
Node.js 設定の名前空間 949
[Procfile] 952
依存関係の設定 953
npm shrinkwrap ファイル 957
プロキシ設定 958
サンプルアプリケーションとチュートリアル 961
Node.js サンプルアプリケーションでの環境の起動 962
次のステップ 963
チュートリアル - Express 963
前提条件 963
Elastic Beanstalk 環境の作成 964
Express を使用するようアプリケーションを更新する 967
Amazon RDS を使用するようにアプリケーションを更新する 970
クリーンアップ 975
チュートリアル - クラスタリング付きの Express 975
前提条件 976
Elastic Beanstalk 環境の作成 976
Express を使用するようアプリケーションを更新する 979
クリーンアップ 992
チュートリアル - Node.js と DynamoDB 993
前提条件 994
Elastic Beanstalk 環境の作成 994
環境内のインスタンスにアクセス許可を追加します 997
サンプルアプリケーションをデプロイする 997
DynamoDB テーブルを作成する 999
アプリケーションの設定ファイルを更新する 1000
高可用性のための環境を設定する 1002
クリーンアップ 1003
次のステップ 1003
データベースを追加します 1004
環境に DB インスタンスを追加 1004
ドライバのダウンロード 1005

データベースへの接続 1006
リソース 1007
PHP のデプロイ 1008
PHP の QuickStart 1008
AWS アカウント 1009
前提条件 1011
ステップ 1: PHP アプリケーションを作成する 1011
ステップ 2: アプリケーションをローカルに実行する 1011
ステップ 3: PHP アプリケーションを初期化してデプロイする 1012
ステップ 4: クラウドアプリケーションを参照する 1012
ステップ 5: アプリケーションを更新して再デプロイする 1012
クリーンアップ 743
次のステップ 1014
PHP プラットフォーム 1014
AWS SDK for PHP のインストール 1016
PHP 8.1 on Amazon Linux 2 に関する考慮事項 1016
PHP 環境の設定 1019
設定の名前空間 1020
依存関係をインストールする 1022
Composer の更新 1023
php.ini の拡張 1024
高度な の例 1025
データベースを追加します 1025
チュートリアル - Laravel 1029
チュートリアル - CakePHP 1038
チュートリアル - Symfony 1046
チュートリアル - HA プロダクション 1052
チュートリアル - HA WordPress 1061
チュートリアル - HA Drupal 1077
Python のデプロイ 1094
Python の QuickStart 1094
AWS アカウント 1095
前提条件 1097
ステップ 1: Python アプリケーションを作成する 1099
ステップ 2: アプリケーションをローカルに実行する 1099
ステップ 3: EB CLI を使用して Python アプリケーションをデプロイする 1100

ステップ 4: Elastic Beanstalk でアプリケーションを実行する 1101
ステップ 5：クリーンアップ 743
AWS アプリケーションの リソース 1101
次のステップ 1102
コンソールでデプロイする 1103
デベロッパー環境 1103
前提条件 1103
仮想環境の使用 1104
Elastic Beanstalk 用の Python プロジェクトの設定 1105
Python プラットフォーム 1106
Python 環境の設定 1107
Python 設定の名前空間 1110
python3 実行可能ファイル 1111
[Procfile] 1112
依存関係の指定 1112
チュートリアル - flask 1115
前提条件 1115
Flask で Python 仮想環境を設定する 1116
Flask アプリケーションを作成する 1117
EB CLI でサイトをデプロイします 1119
クリーンアップ 1122
次のステップ 1123
チュートリアル - Django 1123
前提条件 1124
Python 仮想環境のセットアップと Django のインストール 1124
Django プロジェクトを作成する 1126
Elastic Beanstalk 用に Django アプリケーションを設定する 1128
EB CLI でサイトをデプロイします 1129
アプリケーションの更新 1132
クリーンアップ 1136
次のステップ 1136
データベースを追加します 1136
環境に DB インスタンスを追加 1137
ドライバのダウンロード 1138
データベースへの接続 1139
リソース 1139

Ruby のデプロイ 1140
デベロッパー環境 1140
Ruby のインストール 1141
AWS SDK for Ruby のインストール 1142
IDE またはテキストエディタをインストールする 1142
Ruby プラットフォーム 1143
Ruby 環境の設定 1144
Ruby 設定の名前空間 1147
Gemfile 1148
[Procfile] 1149
チュートリアル - rails 1150
前提条件 1151
Elastic Beanstalk の基本的な知識 1151
Elastic Beanstalk 環境の起動 1152
rails をインストールしてウェブサイトを生成します 1154
rails 設定の構成 1157
アプリケーションをデプロイします 1158
クリーンアップ 1158
次のステップ 1158
チュートリアル - sinatra 1159
前提条件 1159
Elastic Beanstalk 環境の起動 1159
ベーシック sinatra ウェブサイトの書き込み 1162
アプリケーションをデプロイします 1163
クリーンアップ 1164
次のステップ 1164
データベースを追加します 1164
環境に DB インスタンスを追加 1165
アダプタのダウンロード 1166
データベースへの接続 1167
Docker を使用したデプロイ 1168
Docker プラットフォームブランチ 1168
Amazon Linux AMI (AL1) で実行されている廃止されたプラットフォームブランチ 1169
Docker プラットフォームブランチ 1170
Docker の QuickStart 1170
Docker Compose のQuickStart 1178

Docker イメージ設定 1189
ECS マネージドプラットフォームブランチ 1195
ECS マネージド Docker プラットフォームの概要 1196
Elastic Beanstalk によって作成された Amazon ECS リソース 1196
`Dockerrun.aws.json` v2 ファイル 1197
Docker イメージ 1197
失敗したコンテナのデプロイ 1198
ECS ベースの Docker プラットフォームの拡張 1198
Elastic Beanstalk の ECS マネージド Docker 設定 1198
チュートリアル - ECS マネージド Docker 1211
AL2023 上で動作する ECS への移行 1220
イメージリポジトリを使用した認証 1222
Amazon ECR リポジトリ 1222
SSM パラメータストアまたは AWS Secrets Manager 1224
Dockerrun.aws.json ファイル 1227
Amazon ECR パブリックイメージ 1229
環境設定 1230
Docker 環境でのソフトウェアの設定 1231
コンテナ内の環境変数の参照 1232
Docker Compose を使用した環境変数の補間機能の使用 1234
Docker Compose を使用した拡張ヘルスレポート用のログの生成 1235
Docker Compose を使用した Docker コンテナのカスタマイズされたログ記録 1235
Docker イメージ 1237
Docker 環境に対する管理された更新の設定 1237
Docker 設定の名前空間 1238
(Amazon Linux 2 より前の) Amazon Linux AMI での Docker 設定 1239
レガシープラットフォーム 1241
(レガシー) Amazon Linux 2 上で動作する Docker への移行 1241
(レガシー) Docker GlassFish コンテナ 1247
環境のモニタリング 1256
モニタリングコンソール 1256
モニタリンググラフ 1257
モニタリングコンソールのカスタマイズ 1257
CLI によるヘルスのモニタリング 1258
出力の読み取り 1261
インタラクティブヘルスビュー 1263

インタラクティブヘルスビューのオプション 1265
ベーシックヘルスレポート 1265
ヘルスステータスの色 1266
Elastic Load Balancing のヘルスチェック 1267
単一インスタンスおよびワーカー枠環境のヘルスチェック 1268
追加のチェック 1268
Amazon CloudWatch メトリクス 1269
拡張ヘルスレポートおよびモニタリング 1270
Elastic Beanstalk のヘルスエージェント 1272
インスタンスと環境の状態を判断するための要素 1273
ヘルスチェックルールのカスタマイズ 1276
拡張ヘルスレポートのロール 1276
拡張ヘルス認可 1277
拡張ヘルスレポートのイベント 1278
更新、デプロイ、およびスケーリング中の拡張ヘルスレポートの動作 1278
拡張ヘルスレポートの有効化 1279
ヘルスコンソール 1282
状態の色とステータス 1285
インスタンスメトリクス 1287
拡張ヘルスルール 1290
CloudWatch 1295
API ユーザー 1304
拡張ヘルスログ形式 1305
通知とトラブルシューティング 1309
アラームの管理 1311
変更履歴の表示 1313
イベントの表示 1315
コンソール 1315
コマンドライン 1315
インスタンスのモニタリング 1316
インスタンスログの表示 1318
Amazon EC2 インスタンス上のログの場所 1320
Amazon S3 のログの場所 1321
Linux でのログのローテーション設定 1322
デフォルトのログタスク設定の拡張 1322
Amazon CloudWatch Logs へのログファイルのストリーミング 1325

AWS サービスの統合 1327
アーキテクチャの概要 1327
CloudFront 1328
CloudTrail 1328
CloudTrail 履歴の Elastic Beanstalk 情報 1329
Elastic Beanstalk ログファイルエントリの概要 1330
CloudWatch 1331
CloudWatch Logs 1332
CloudWatch Logs へのログストリーミングの前提条件 1333
Elastic Beanstalk が CloudWatch Logs を設定する方法 1333
CloudWatch Logs へのインスタンスログのストリーミング 1339
CloudWatch Logs 統合のトラブルシューティング 1341
環境ヘルスのストリーミング 1342
EventBridge 1345
EventBridge を使用した Elastic Beanstalk リソースの監視 1346
Elastic Beanstalk イベントパターンの例 1349
Elastic Beanstalk イベントの例 1351
Elastic Beanstalk イベントフィールドマッピング 1352
AWS Config 1355
セットアップ AWS Config 1356
Elastic Beanstalk リソースを記録する AWS Config ように を設定する 1356
AWS Config コンソールでの Elastic Beanstalk 設定の詳細の表示 1357
AWS Config ルールを使用した Elastic Beanstalk リソースの評価 1357
DynamoDB 1358
ElastiCache 1359
Amazon EFS 1360
設定ファイル 1360
暗号化されたファイルシステム 1361
サンプルアプリケーション 1361
ファイルシステムのクリーンアップ 1362
IAM 1362
インスタンスプロファイル 1363
サービス役割 1368
サービスにリンクされたロールの使用 1376
ユーザーポリシー 1388
ARN 形式 1396

リソースと条件 1398
タグベースのアクセス制御 1446
管理ポリシーの例 1451
リソース固有のポリシーの例 1455
環境間の S3 バケットアクセス 1466
Amazon RDS 1468
デフォルトの VPC の Amazon RDS 1470
Amazon RDS 認証情報とシークレットマネージャー 1475
外部 Amazon RDS インスタンスのクリーンアップ 1475
Amazon S3 1476
Elastic Beanstalk Amazon S3 カスタマーアカウントバケット 1476
Elastic Beanstalk Amazon S3 カスタマーアカウントバケットの内容 1476
Elastic Beanstalk Amazon S3 バケット内のオブジェクトの削除 1477
Elastic Beanstalk Amazon S3 バケットの削除 1478
Secrets Manager と Systems Manager パラメータストア 1479
環境変数へのシークレットの取得 1480
必要な IAM 許可 1487
Secrets Manager と Systems Manager パラメータストアの使用 1489
シークレットと環境変数のトラブルシューティング 1491
Amazon VPC 1492
パブリック VPC 1494
パブリック/プライベート VPC 1495
プライベート VPC 1496
踏み台ホスト 1498
Amazon RDS 1502
VPC エンドポイント 1506
VPC エンドポイントポリシー 1511
セキュリティ 1522
データ保護 1523
データの暗号化 1524
インターネットのプライバシー 1525
Identity and Access Management 1525
AWS マネージドポリシー 1526
ログ記録とモニタリング 1540
拡張ヘルスレポート 1540
Amazon EC2 インスタンス名 1540

環境の通知 1541
Amazon CloudWatch アラーム 1541
AWS CloudTrail ログ 1541
AWS X-Ray デバッグ 1541
コンプライアンス検証 1541
耐障害性 1542
インフラストラクチャセキュリティ 1543
責任共有モデル 1543
セキュリティに関するベストプラクティス 1544
予防的セキュリティのベストプラクティス 1544
セキュリティ問題の検出ベストプラクティス 1545
アクセス許可 1547
必要なロール 1547
オプションのポリシーとロール 1547
サービスロール 1548
AWSElasticBeanstalkEnhancedHealth 1369
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy 1369
インスタンスプロファイル 1550
マネージドポリシー 1550
EC2 インスタンスプロファイルの作成 1551
ユーザーポリシー 1551
チュートリアルおよびサンプル 1553
IIS アプリケーションの移行 1556
前提条件 1557
移行用語集 1559
Windows、IIS、および .NET の用語 1559
Elastic Beanstalk 用語 1563
Python 用語 1567
IIS の基本的な移行 1569
IIS 環境の探索 1569
移行の準備 1570
最初の移行 1571
移行の制御 1572
進行状況の監視 1572
移行の検証 1573
移行アーティファクトの管理 1574

ネットワーク構成 1574
VPC の構成 1574
ポート設定を使用したマルチサイトデプロイ 1579
共有設定と依存関係 1581
ベストプラクティス 1581
トラブルシューティング 1582
セキュリティ設定 1583
インスタンスプロファイル設定 1583
サービスロールの管理 1584
セキュリティグループの構成 1585
SSL 証明書の統合 1586
Windows 認証 1586
ベストプラクティスとトラブルシューティング 1587
IIS から Elastic Beanstalk への移行マッピング 1589
Elastic Beanstalk の IIS サイトとアプリケーション 1589
仮想ディレクトリとアプリケーションパスの管理 1591
URL 書き換えとアプリケーションリクエストルーティング (ARR) 1592
移行アーティファクト構造 1592
高度な移行シナリオ 1594
アプリケーションリクエストルーティング (ARR) を使用したマルチサイト移行 1594
ホストベースのルーティングを使用した ARR なしのマルチサイト移行 1597
仮想ディレクトリ管理 1599
カスタムアプリケーションプール設定 1603
以前のバージョンのデプロイ 1605
トラブルシューティングと診断 1606
EC2 キーペアを環境に関連付ける 1606
ログへのアクセス 1607
クライアント側のアーティファクトへのアクセス 1607
環境の状態のモニタリング 1607
EC2 パフォーマンスの最適化 1608
EBS ボリューム設定 1608
一般的な の問題と解決策 1609
サポート情報 1610
移行オプション: EB CLI と MGN 1610
各移行オプションを使用するタイミング 1611
移行ワークフローの比較 1612

結論 1613
トラブルシューティング 1614
Systems Manager ツールの使用 1615
一般的な質問、または機能要望 1616
シークレットの環境変数 1617
環境の作成 1618
デプロイ 1618
健康 1619
設定 1620
Docker 1620
よくある質問 1621
デプロイのトラブルシューティング 1622
デプロイエラー 1623
リソース 1627
サンプルアプリケーション 1628
AWS SDK for Java 1628
AWS SDK for .NET 1629
AWS Toolkit for Visual Studio 1629
AWS Node.js での SDK for JavaScript 1629
AWS SDK for PHP 1630
AWS SDK for Python (Boto) 1630
AWS SDK for Ruby 1630
アーカイブされたコンテンツ 1631
カスタムプラットフォーム (廃止) 1631
ドキュメント履歴 1649
..... mdclv

とは AWS Elastic Beanstalk

Elastic Beanstalk を使用すると、サポートされているさまざまなプラットフォームでウェブアプリケーションを AWS クラウドにデプロイできます。アプリケーションを構築してデプロイします。Elastic Beanstalk は Amazon EC2 インスタンスのプロビジョニング、負荷分散の設定、ヘルスモニタリングの設定、環境の動的なスケーリングを行います。

ウェブサーバー環境に加えて、Elastic Beanstalk には Amazon SQS キューからのメッセージを処理するために使用できるワーカー環境も用意されており、非同期タスクや長時間実行されるタスクに役立ちます。詳細については、「Elastic Beanstalk ワーカー環境」を参照してください。

サポートされているプラットフォーム

Elastic Beanstalk は、Go、Java、.NET、Node.js、PHP、Python、Ruby で開発されたアプリケーションをサポートします。Elastic Beanstalk は Docker コンテナもサポートしており、独自のプログラミング言語とアプリケーションの依存関係を選択できます。アプリケーションをデプロイすると、Elastic Beanstalk は選択したサポートされているプラットフォームバージョンを構築し、Amazon EC2 インスタンスなどの 1 つ以上の AWS リソースを AWS アカウントにプロビジョニングしてアプリケーションを実行します。

Elastic Beanstalk コンソール、 AWS Command Line Interface （AWS CLI）、または Elastic Beanstalk 専用に設計された高レベルのコマンドラインツールである EB CLI を使用して、Elastic Beanstalk とやり取りできます。

Amazon EC2 インスタンスのフリートのサイズ変更やアプリケーションのモニタリングなど、ほとんどのデプロイタスクは、Elastic Beanstalk ウェブインターフェイス (コンソール) から直接実行できます。

Elastic Beanstalk を使用してサンプルウェブアプリケーションをデプロイする方法の詳細については、「」を参照してくださいElastic Beanstalk の使用を開始する方法について説明します。。

アプリケーションのデプロイワークフロー

Elastic Beanstalk を使用するには、アプリケーションを作成し、アプリケーションソースバンドルを Elastic Beanstalk にアップロードします。次に、アプリケーションに関する情報を入力すると、Elastic Beanstalk は自動的に環境を起動し、コードの実行に必要な AWS リソースを作成して設定します。

アプリケーションを作成してデプロイし、環境を起動したら、環境を管理し、新しいアプリケーションバージョンをデプロイできます。メトリクス、イベント、環境ステータスなど、アプリケーションに関する情報は、Elastic Beanstalk コンソール、APIs、コマンドラインインターフェイスから入手できます。

次の図は、Elastic Beanstalk ワークフローを示しています。

料金

Elastic Beanstalk に対する追加料金はありません。支払いは、アプリケーションが消費する基盤となる AWS リソースに対してのみ行われます。料金の詳細については、Elastic Beanstalk サービスの詳細ページを参照してください。

次のステップ

Elastic Beanstalk の使用を開始するには入門チュートリアル、チュートリアル をお勧めします。このチュートリアルでは、サンプルの Elastic Beanstalk アプリケーションを作成、表示、更新する手順を説明します。

Elastic Beanstalk の使用を開始する方法について説明します。

Elastic Beanstalk を使用すると、ウェブアプリケーションとサービスをデプロイ、モニタリング、スケーリングできます。通常、コードをローカルで開発し、Amazon EC2 サーバーインスタンスにデプロイします。これらのインスタンスは環境とも呼ばれ、 AWS コンソールまたはコマンドラインを介してアップグレードできるプラットフォームで実行されます。

開始するには、コンソールから直接構築済みのサンプルアプリケーションをデプロイすることをお勧めします。次に、ローカルで開発し、 のコマンドラインからデプロイする方法について説明しますthe section called “PHP の QuickStart”。

Elastic Beanstalk の使用には料金はかかりませんが、このチュートリアルの過程で作成した AWS リソースは、最後に削除するまで標準料金が適用されます。通常、合計料金は 1 ドル未満です。 使用料を最小限に抑える方法については、「AWS 無料利用枠」を参照してください。

このチュートリアルを完了すると、Amazon EC2 インスタンスで実行されている環境を使用して Elastic Beanstalk アプリケーションを作成、設定、デプロイ、更新、モニタリングするための基本を理解できます。

推定所要時間: 35 ~ 45 分

構築する内容

最初の Elastic Beanstalk アプリケーションは、PHP マネージドプラットフォームで PHP サンプルを実行する単一の Amazon EC2 環境で構成されます。

Elastic Beanstalk アプリケーション

> Elastic Beanstalk アプリケーションは、Elastic Beanstalk コンポーネント用のコンテナです。これには、アプリケーションコードが Elastic Beanstalk によって提供および管理されるプラットフォームで実行される環境や、ユーザーが提供するカスタムコンテナで実行される環境が含まれます。

環境

Elastic Beanstalk 環境は、Amazon EC2 インスタンスを含む、一緒に実行される AWS リソースのコレクションです。環境を作成すると、Elastic Beanstalk は必要なリソースを AWS アカウントにプロビジョニングします。

プラットフォーム

プラットフォームは、オペレーティングシステム、プログラミング言語ランタイム、ウェブサーバー、アプリケーションサーバー、および追加の Elastic Beanstalk コンポーネントの組み合わせです。Elastic Beanstalk は、管理されたプラットフォームを提供するか、コンテナに独自のプラットフォームを提供できます。

Elastic Beanstalk は、さまざまなプログラミング言語、アプリケーションサーバー、および Docker コンテナ用のプラットフォームをサポートします。環境を作成するときは、プラットフォームを選択する必要があります。プラットフォームはアップグレードできますが、環境のプラットフォームを変更することはできません。

> **プラットフォームの切り替え**
>
> プログラミング言語を変更する必要がある場合は、別のプラットフォームで新しい環境を作成して切り替える必要があります。

ステップ 1 - アプリケーションを作成する

サンプルアプリケーションを作成するには、[Create application] (アプリケーションの作成) コンソールウィザードを使用します。Elastic Beanstalk アプリケーションを作成し、その中に環境を起動します。

注意: 環境は、アプリケーションコードの実行に必要な AWS リソースのコレクションです。

アプリケーションを作成する方法

1. Elastic Beanstalk コンソールで開きます。
2. [Create application] を選択します。
3. [Application name] (アプリケーション名) に **`getting-started-app`** と入力します。

コンソールには、アプリケーションを作成し、環境を設定するための 6 ステップのプロセスが用意されています。このクイックスタートでは、最初の 2 つのステップに集中するだけで済みます。次にスキップして、アプリケーションと環境を確認して作成できます。

環境を設定するには

1. 環境情報で、環境名に「」と入力します**gs-app-web-env**。
2. プラットフォーム で、PHP プラットフォームを選択します。
3. アプリケーションコードとプリセットで、デフォルト (サンプルアプリケーションと単一インスタンス) を受け入れ、次へを選択します。

サービスアクセスを設定するには

次に、2 つのロールが必要です。サービスロールを使用すると、Elastic Beanstalk は EC2 インスタンスをモニタリングし、環境のプラットフォームをアップグレードできます。EC2 インスタンスプロファイルロールは、ログの書き込みや他の サービスとの対話などのタスクを許可します。

サービスロールを作成するには

1. サービスロール で、ロールの作成 を選択します。
2. [信頼できるエンティティタイプ] で、[AWS サービス] を選択してください。
3. ユースケースでは、Elastic Beanstalk – Environment を選択します。
4. [次へ] を選択します。
5. アクセス許可ポリシーに以下が含まれていることを確認し、次へを選択します。
   - `AWSElasticBeanstalkEnhancedHealth`
   - `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy`
6. [ロールの作成] を選択してください。
7. サービスアクセスの設定タブに戻り、リストを更新して、新しく作成したサービスロールを選択します。

EC2 インスタンスプロファイルを作成するには

1. [ロールの作成] を選択してください。
2. [信頼できるエンティティタイプ] で、[AWS サービス] を選択します。
3. ユースケースで、Elastic Beanstalk – Compute を選択します。

4. [次へ] を選択します。
5. アクセス許可ポリシーに以下が含まれていることを確認し、次へを選択します。
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`
6. [ロールの作成] を選択してください。
7. サービスアクセスの設定タブに戻り、リストを更新して、新しく作成した EC2 インスタンスプロファイルを選択します。

アプリケーションの設定と作成を完了するには

1. EC2 キーペアをスキップします。

   コンソールから Amazon EC2 インスタンスに接続するその他の方法を示します。
2. 「レビューにスキップ」を選択して、いくつかのオプションステップを移動します。

   オプションの手順: ネットワーク、データベース、スケーリングパラメータ、更新の高度な設定、モニタリング、ログ記録。
3. 選択の概要を示すレビューページで、送信を選択します。

> お疲れ様でした。
>
> アプリケーションを作成し、環境を設定しました。次に、リソースがデプロイされるのを待つ必要があります。

ステップ 2 - アプリケーションをデプロイする

アプリケーションを作成すると、Elastic Beanstalk が環境をセットアップします。座って待つだけで済みます。

初期デプロイでは、リソースの作成に最大 5 分かかる場合があります。変更のみがスタックにデプロイされるため、更新にかかる時間が短くなります。

サンプルアプリケーションを作成すると、Elastic Beanstalk によって次の S リソースが作成されます。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するように設定された Amazon EC2 仮想マシン。

  各プラットフォームは、特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするために、さまざまなソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または nginx をリバースプロキシとして使用して、ウェブトラフィックをウェブアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。Amazon EC2 インスタンスに接続して、設定とログを表示できます。
- インスタンスセキュリティグループ – Amazon EC2 セキュリティグループが作成され、ポート 80 での受信リクエストが許可されるため、ロードバランサーのインバウンドトラフィックはウェブアプリケーションに到達できます。
- Amazon S3 バケット – ソースコード、ログ、その他のアーティファクトのストレージの場所。
- Amazon CloudWatch アラーム – インスタンスの負荷をモニタリングし、必要に応じてスケールアップまたはスケールダウンするために 2 つの CloudWatch アラームが作成されます。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して 環境にリソースをデプロイし、設定を変更します。リソース定義テンプレートは AWS CloudFormation コンソールで表示できます。
- ドメイン名 – ウェブアプリにルーティングするドメイン名。
  `subdomain.region.elasticbeanstalk.com`

Elastic Beanstalk は、アプリケーションを作成し、環境を起動し、アプリケーションバージョンを作成し、コードを環境にデプロイします。プロセス中、コンソールは進行状況を追跡し、イベントタブにイベントステータスを表示します。

すべてのリソースがデプロイされると、環境の状態は に変わります**Ok**。

> アプリケーションの準備ができました。
>
> アプリケーションの状態が に変わったら**Ok**、ウェブアプリケーションのウェブサイトを参照できます。

ステップ 3 - Elastic Beanstalk 環境を調べる

コンソールの環境の概要ページから、デプロイされたアプリケーション環境の探索を開始します。

環境とアプリケーションを表示するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境に移動を選択してアプリケーションを参照します。

   （ドメインにリストされている URL リンクを選択してアプリケーションを参照することもできます）。

   接続は HTTP (HTTPS ではない) になるため、ブラウザに警告が表示されることがあります。

Elastic Beanstalk コンソールに戻ると、上部に環境の概要と、名前、ドメイン URL、現在のヘルスステータス、実行中のバージョン、アプリケーションが実行されているプラットフォームなど、環境に関する最上位の情報が表示されます。実行中のバージョンとプラットフォームは、現在デプロイされているアプリケーションのトラブルシューティングに不可欠です。

概要ペインの後に、イベントタブに最近の環境アクティビティが表示されます。

Elastic Beanstalk が AWS リソースを作成し、アプリケーションを起動している間、環境は `Pending`状態になります。起動イベントに関するステータスメッセージは、イベント のリストに継続的に追加されます。

環境のドメインは、デプロイされたウェブアプリケーションの URL です。左側のナビゲーションペインの「環境に移動」でもドメインに移動します。同様に、左側のナビゲーションペインには、さまざまなタブに対応するリンクがあります。

左側のナビゲーションペインの設定リンクに注意してください。環境設定オプション値の概要がカテゴリ別にグループ化されて表示されます。

> **環境設定**
>
> 左側のナビゲーションペインの設定リンクを書き留めます。サービスロール、ネットワーク、データベース、スケーリング、マネージドプラットフォームの更新、メモリ、ヘルスモ

ニタリング、ローリングデプロイ、ログ記録など、詳細な環境設定を表示および編集できます。

さまざまなタブには、環境に関する詳細情報が含まれています。

- イベント – 環境内のリソースの Elastic Beanstalk サービスおよびその他の サービスからの情報とエラーメッセージの更新リストを表示します。
- Health – アプリケーションを実行している Amazon EC2 インスタンスのステータスと詳細なヘルス情報を表示します。
- [Logs] (ログ) - 環境内の Amazon EC2 からログを取得してダウンロードします。ログ全体または最近のアクティビティを取得できます。取得したログは 15 分間使用できます。
- モニタリング – 平均レイテンシーや CPU 使用率など、環境の統計を表示します。
- アラーム – 環境メトリクス用に設定されたアラームを表示および編集します。
- マネージド更新 – 今後および完了したマネージドプラットフォームの更新とインスタンスの置き換えに関する情報を表示します。
- タグ – 環境に適用されるキーと値のペアを表示および編集します。

> **Note**
> コンソールナビゲーションペインのリンクには、対応するタブが表示されます。

ログによるトラブルシューティング

予期しない動作のトラブルシューティングやデプロイのデバッグを行うには、環境のログを確認することをお勧めします。

Elastic Beanstalk コンソールのログタブで、すべてのログファイルの 100 行をリクエストできます。または、Amazon EC2 インスタンスに直接接続し、ログをリアルタイムでテーリングすることもできます。

ログをリクエストするには (Elastic Beanstalk コンソール)

1. Elastic Beanstalk コンソールで環境に移動します。
2. ログタブまたは左ナビゲーションを選択し、リクエストログを選択します。

3. 最後の 100 行を選択します。
4. ログを作成したら、ダウンロードリンクを選択してブラウザでログを表示します。

ログで、 ログを検索し、nginx アクセスログの ディレクトリを書き留めます。

Amazon EC2 への接続を有効にするポリシーを追加する

接続する前に、Session Manager を使用して Amazon EC2 への接続を有効にするポリシーを追加する必要があります。

1. [IAM console] (IAM コンソール) に入ります。
2. **`aws-elasticbeanstalk-ec2-role`** ロールを検索して選択します。
3. アクセス許可を追加を選択し、ポリシーをアタッチします。
4. 次のテキストで始まるデフォルトポリシーを検索し**`AmazonSSMManagedEC2Instance`**、ロールに追加します。

Session Manager を使用して Amazon EC2 に接続するには

1. Amazon EC2 コンソールに移動します。
2. インスタンスを選択し、`gs-app-web-env`インスタンスを選択します。
3. Connect を選択し、Session Manager を選択します。
4. [接続] を選択します。

インスタンスに接続したら、bash シェルを起動し、ログをテーリングします。

1. **`bash`** コマンドを実行します。
2. **`cd /var/log/nginx`** コマンドを実行します。
3. **`tail -f access.log`** コマンドを実行します。
4. ブラウザで、アプリケーションドメイン URL に移動します。更新します。

> おめでとうございます。接続されました。
> ページを更新するたびに、インスタンスの更新にログエントリが表示されます。

> **接続ボタンが機能しませんか？**
>
> 接続ボタンが使用できない場合は、IAM に戻り、必要なポリシーをロールに追加したことを確認します。

ステップ 4 - アプリケーションを更新する

最終的には、アプリケーションを更新します。環境で現在実行中の他の更新オペレーションがなければ、アプリケーションの新しいバージョンはいつでもデプロイ可能です。

このチュートリアル開始時のアプリケーションバージョンは、サンプルアプリケーションと呼ばれます。

アプリケーションバージョンを更新するには

1. 次の PHP サンプルアプリケーションをダウンロードします。

   PHP – php-v2.zip
2. Elastic Beanstalk 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。
3. ナビゲーションペインで、[環境] を選択し、リストから環境の名前を選択します。
4. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
5. [ファイルを選択] を選択し、ダウンロードしたサンプルアプリケーションソースバンドルをアップロードします。

   コンソールは、バージョンラベルに新しい一意のラベルを自動的に入力し、末尾の整数を自動的に増分します。独自のバージョンラベルを選択する場合は、一意であることを確認してください。
6. [デプロイ] を選択します。

Elastic Beanstalk がファイルを Amazon EC2 インスタンスにデプロイしている間、[環境概要] ページでデプロイステータスを表示できます。アプリケーションバージョンがアップデートされている間、環境の [ヘルス] ステータスは灰色になります。デプロイが完了すると、Elastic Beanstalk によってアプリケーションのヘルスチェックが実行されます。アプリケーションがヘルスチェックに応答すると、正常と見なされ、ステータスは緑色に戻ります。環境概要には、新しい [実行バージョン] に [バージョンラベル] として指定した名前が表示されます。

Elastic Beanstalk により、新しいアプリケーションバージョンがアップロードされ、アプリケーションバージョンのテーブルにも追加されます。テーブルを表示するには、ナビゲーションペインの [getting-started-app] で [Application versions] (アプリケーションバージョン) を選択します。

> 更新に成功しました。
>
> ブラウザを更新すると、更新された「v2」メッセージが表示されます。
> ソースを自分で編集する場合は、ソースバンドルを解凍、編集、再圧縮します。macOS では、-X を使用して php ディレクトリ内から次のコマンドを使用して、追加のファイル属性を除外します。
> `zip -X -r ../php-v2.zip .`

ステップ 5 - アプリケーションをスケールする

お使いのアプリケーションに適合するよう、環境をカスタマイズすることができます。例えば、計算量の多いアプリケーションがある場合、アプリケーションを実行する Amazon Elastic Compute Cloud (Amazon EC2) インスタンスの種類を変更できます。設定の変更を適用するために、Elastic Beanstalk は環境の更新を実行します。

一部の簡単な設定の変更はすぐに反映されます。一部の変更では、 AWS リソースを削除して再作成する必要があります。これには数分かかる場合があります。構成設定を変更すると、Elastic Beanstalk により、アプリケーションのダウンタイムが発生する可能性について警告が表示されます。

容量設定を増やす

この構成変更の例では、環境の容量設定を編集します。Auto Scaling グループに 2 つから 4 つの Amazon EC2 インスタンスを持つ負荷分散型のスケーラブルな環境を設定し、変更が行われたことを確認します。Elastic Beanstalk は Amazon EC2 インスタンスを作成し、最初に作成した 1 つのインスタンスに追加します。次に、Elastic Beanstalk は両方のインスタンスを環境内のロードバランサーに関連付けます。その結果、アプリケーションの応答性が向上し、可用性が向上します。

環境の容量を変更するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。

3. ナビゲーションペインで、[設定] を選択します。
4. [Instance traffic and scaling] (インスタンスのトラフィックおよびスケーリング) 設定カテゴリで、[Edit] (編集) を選択します。
5. [Instances] (インスタンス) セクションを折りたたむと、[Capacity] (容量) セクションが見やすくなります。[Auto Scaling group] (Auto Scaling グループ) で、[Environment type] (環境タイプ) を [Load balanced] (負荷分散) に変更します。
6. インスタンス行で、最小 を に**2**、最大 を に変更します**4**。
7. ページの最下部で [適用] を選択し変更を保存します。

   更新によって現在のインスタンスがすべて置き換えられることが警告された場合。確認 を選択します。

環境の更新には数分程かかります。イベントのリストにいくつかの更新が表示されます。イベントをモニタリングする 環境に新しい設定を正常にデプロイしました。

容量の増加を検証する

環境の更新が完了し、環境の準備が完了すると、Elastic Beanstalk は新しい最小容量設定を満たすために 2 番目のインスタンスを自動的に起動します。

増加した容量を確認するには

1. タブリストまたは左側のナビゲーションペインからヘルスを選択します。
2. 拡張インスタンスのヘルスセクションを確認します。

> **スケールアップしたばかりです。**
>
> 2 つの Amazon EC2 インスタンスでは、環境容量が 2 倍になり、数分しかかかりませんでした。

Elastic Beanstalk 環境のクリーンアップ

使用していないサービスに対して課金されないようにするには、すべてのアプリケーションバージョンを削除し、環境を終了してください。これにより、環境が作成した AWS リソースも削除されます。

アプリケーションおよび関連するすべてのリソースを削除するには

1. すべてのアプリケーションバージョンを削除します。
   a. Elastic Beanstalk 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. ナビゲーションペインで、[アプリケーション] を選択し、[getting-started-app] を選択します。
   c. ナビゲーションペインで、アプリケーション名を見つけ、[アプリケーションバージョン] を選択します。
   d. [アプリケーションバージョン] ページで、削除するすべてのアプリケーションバージョンを選択します。
   e. [アクション] を選択し、[削除] を選択します。
   f. [Amazon S3 からのバージョンの削除] をオンにします。
   g. [削除] を選択してから、[完了] を選択します。
2. 環境を終了します。
   a. ナビゲーションペインで、[gettinging-started-app] を選択してから、環境リストで [GettingStartedApp-env] を選択します。
   b. [アクション]、[環境の終了] の順に選択します。
   c. 環境名を入力し、GettingStartedApp-env の終了を確認した後に、[Terminate] (終了) を選択します。
3. getting-started-app アプリケーションを削除します。
   a. ナビゲーションペインで、[getting-started-app] を選択します。
   b. [アクション] を選択してから、[アプリケーションの削除] を選択します。
   c. アプリケーション名を入力して [getting-started-app] を削除することを確認し、[削除] を選択します。

> **お疲れ様でした。**
>
> サンプルアプリケーションを AWS クラウドに正常にデプロイし、新しいバージョンをアップロードし、その設定を変更して 2 番目の Auto Scaling インスタンスを追加し、リソースをクリーンアップしました AWS 。

次のステップ

eb コマンドラインツールを使用して Elastic Beanstalk へのコードのデプロイを自動化する方法については、「」を続行することをお勧めしますthe section called “PHP の QuickStart”。

次に、HTTPS 接続の設定方法を確認します。「」を参照してくださいthe section called “HTTPS”。

Elastic Beanstalk を管理するための EB コマンドラインインターフェイス (EB CLI) の設定

EB CLI は、 で環境を作成、更新、モニタリングするためのインタラクティブなコマンドを提供するコマンドラインインターフェイスです AWS Elastic Beanstalk。EB CLI オープンソースプロジェクトは Github にあります。 aws/aws-elastic-beanstalk-cli

EB CLI をインストールし、プロジェクトディレクトリを設定したら、1 つのコマンドで環境を作成できます。

```
$ eb create <my-beanstalk-environment>
```

セットアップスクリプトを使用して をインストールすることをお勧めします。方法については、「」を参照してくださいthe section called “EB CLI のインストール”。

AWS CLI は、低レベルの Elastic Beanstalk APIs への直接アクセスを提供します。強力ですが、詳細であり、EB CLI よりも優先されません。たとえば、 を使用して環境を作成するには、次の一連のコマンド AWS CLI が必要です。

```
$ aws elasticbeanstalk check-dns-availability \
    --cname-prefix my-cname
$ aws elasticbeanstalk create-application-version \
    --application-name my-application \
    --version-label v1 \
    --source-bundle S3Bucket=amzn-s3-demo-bucket,S3Key=php-proxy-sample.zip
$ aws elasticbeanstalk create-environment \
    --cname-prefix my-cname \
    --application-name my-app \
    --version-label v1 \
    --environment-name my-env \
    --solution-stack-name "64bit Amazon Linux 2023 v4.5.0 running Ruby 3.4"
```

セットアップスクリプトを使用して EB CLI をインストールする (推奨)

> インストーラスクリプトをお勧めします。
>
> インストーラスクリプトを使用して EB CLI とその依存関係を設定し、他の Python パッケージとの潜在的な競合を防ぐことをお勧めします。

前提条件: Git、Python、および virtualenv

インストーラスクリプトをダウンロードして使用するには

1. リポジトリをクローン作成します。

```
git clone https://github.com/aws/aws-elastic-beanstalk-cli-setup.git
```

2. EB CLI をインストールまたはアップグレードします。

   Bash または Zsh の macOS / Linux

```
python ./aws-elastic-beanstalk-cli-setup/scripts/ebcli_installer.py
```

   PowerShell または コマンドウィンドウの Windows

```
python .\aws-elastic-beanstalk-cli-setup\scripts\ebcli_installer.py
```

3. EB CLI が正しくインストールされたことを確認します。

```
$ eb --version
EB CLI 3.21.0 (Python 3.12)
```

インストール手順の詳細については、GitHub の `aws/aws-elastic-beanstalk-cli-setup`リポジトリを参照してください。

「EB CLI の手動インストール」

Python pipパッケージとその依存関係のインストール、アップグレード、削除を提供する Python のパッケージマネージャーを使用して、Linux、macOS、Windows に EB CLI をインストールできます。

> **インストーラスクリプトをお勧めします。**
>
> を使用して EB CLI のインストール EB CLI を設定し、依存関係の競合を防ぐことをお勧めします。

前提条件 - サポートされているバージョンの Python がインストールされている必要があります。Python ウェブサイトの Python ダウンロードページからダウンロードできます。

EB CLI をインストールするには (手動で)

1. 以下のコマンドを実行してください。

```
$ pip install awsebcli --upgrade --user
```

   `--upgrade` オプションでは、すでにインストールされている要件をアップグレードするよう `pip` に指示します。`--user` オプションでは、オペレーティングシステムによって使用されるライブラリの変更を避けるために、ユーザーディレクトリのサブディレクトリにプログラムをインストールするよう `pip` に指示します。

   > **問題のトラブルシューティング**
   >
   > `pip` で EB CLI のインストールを試みたときに依存関係の問題が発生した場合は、仮想環境に EB CLI をインストールしてツールとその依存関係を隔離するか、通常使用しているものと異なるバージョンの Python を使用できます。

2. `PATH` 変数に実行可能ファイルへのパスを追加します。
   - Linux および macOS の場合:

     Linux – `~/.local/bin`

     macOS – `~/Library/Python/3.12/bin`

PATH 変数を変更するには (Linux、Unix、macOS):

a. ユーザーフォルダーでシェルのプロファイルスクリプトを見つけます。現在使用しているシェルがわからない場合は、`echo $SHELL` を実行します。

```
$ ls -a ~
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
```

- Bash – `.bash_profile`、`.profile`、または `.bash_login`。
- Zsh - `.zshrc`
- Tcsh – `.tcshrc`、`.cshrc`、または `.login`。

b. プロファイルスクリプトにエクスポートコマンドを追加します。次の例では、現在の PATH 変数に *`LOCAL_PATH`* で示されるパスを追加します。

```
export PATH=LOCAL_PATH:$PATH
```

c. 最初のステップで説明したプロファイルスクリプトを現在のセッションにロードします。次の例では、*`PROFILE_SCRIPT`* で表されるプロファイルスクリプトをロードします。

```
$ source ~/PROFILE_SCRIPT
```

- Windows の場合:

Python 3.12 – `%USERPROFILE%\AppData\Roaming\Python\Python312\Scripts`

Python の以前のバージョン – `%USERPROFILE%\AppData\Roaming\Python\Scripts`

PATH 変数を変更するには (Windows):

a. Windows キーを押し、**`environment variables`** と入力します。

b. [Edit environment variables for your account] (アカウントの環境変数を編集する) を選択します。

c. PATH を選択して、編集 を選択します。

d. セミコロンで区切って、[Variable value] フィールドにパスを追加します。例: *`C:\item1\path`*;***`C:\item2\path`***

e. [OK] を 2 回選択して、新しい設定を適用します。

f. 実行中のコマンドプロンプトウィンドウを閉じ、コマンドプロンプトウィンドウを再度開きます。

3. eb --version を入力することで、EB CLI が正しくインストールされたことを確認します。

```
$ eb --version
EB CLI 3.21.0 (Python 3.12)
```

EB CLI は、最新の Elastic Beanstalk 機能をサポートする機能を追加するために、定期的に更新されます。EB CLI を最新バージョンに更新するには、インストールコマンドを再び実行します。

```
$ pip install awsebcli --upgrade --user
```

EB CLI をアンインストールする必要がある場合は、`pip uninstall` を使用します。

```
$ pip uninstall awsebcli
```

仮想環境に EB CLI をインストールする

仮想環境に EB CLI をインストールすることで、他の `pip` パッケージとのバージョンに関する要件の競合を回避できます。

仮想環境に EB CLI をインストールするには

1. まず、 `virtualenv`で をインストールします`pip`。

```
$ pip install --user virtualenv
```

2. 仮想環境を作成します。

```
$ virtualenv ~/eb-ve
```

デフォルト以外の Python 実行可能ファイルを使用するには、`-p` オプションを指定します。

```
$ virtualenv -p /usr/bin/python3.12 ~/eb-ve
```

3. 仮想環境をアクティブ化します。

Linux、Unix、または macOS

```
$ source ~/eb-ve/bin/activate
```

Windows

```
$ %USERPROFILE%\eb-ve\Scripts\activate
```

4. EB CLI のインストール。

```
(eb-ve)$ pip install awsebcli --upgrade
```

5. EB CLI が正しくインストールされたことを確認します。

```
$ eb --version
EB CLI 3.23.0 (Python 3.12)
```

`deactivate` コマンドを使用して、仮想環境を終了できます。新しいセッションを開始するたびに、アクティベーションコマンドを再度実行します。

最新バージョンにアップグレードするには、インストールコマンドを再び実行します。

```
(eb-ve)$ pip install awsebcli --upgrade
```

homebrew で EB CLI をインストールする

EB CLI の最新バージョンは、通常、`pip` で表示されてから数日後に `Homebrew` で使用できるようになります。

> ⚠ インストーラスクリプトをお勧めします。
>
> を使用して EB CLI のインストール EB CLI を設定し、依存関係の競合を防ぐことをお勧めします。

`Homebrew` を使用して EB CLI をインストールするには

1. `Homebrew` が最新バージョンであることを確認します。

```
$ brew update
```

2. `brew install awsebcli` を実行します。

```
$ brew install awsebcli
```

3. EB CLI が正しくインストールされたことを確認します。

```
$ eb --version
EB CLI 3.21.0 (Python 3.12)
```

EB CLI の設定

EB CLI をインストールしたら、 を実行してプロジェクトディレクトリと EB CLI を設定する準備が整いますeb init。次の例は、 という名前のプロジェクトフォルダでeb init初めて を実行するときの設定手順を示していますeb。

EB CLI プロジェクトを初期化する

1. まず、EB CLI にはリージョンの選択がプロンプトされます。任意のリージョンを選択します。

```
~/eb $ eb init
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : Europe (Ireland)
5) eu-central-1 : Europe (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
...
(default is 3): 3
```

2. プロンプトが表示されたら、EB CLI がリソースを管理できるように、アクセスキーとシークレットキーを指定します。アクセスキーは AWS Identity and Access Management コンソールで作成されます。キーがない場合には、https://docs.aws.amazon.com/general/latest/gr/getting-aws-sec-creds.htmlの「Amazon Web Services 全般のリファレンス How Do I Get Security Credentials?」を参照してください。

```
You have not yet set up your credentials or your credentials are incorrect.
You must provide your credentials.
(aws-access-id): AKIAJOUAASEXAMPLE
(aws-secret-key): 5ZRIrtTM4ciIAvd4EXAMPLEDtm+PiPSzpoK
```

3. Elastic Beanstalk アプリケーションは、単一のウェブアプリケーションに関連付けられた一連のアプリケーションバージョン (ソース) 、環境、および保存された設定を含むリソースです。EB CLI を使って Elastic Beanstalk にソースコードをデプロイするたびに、新しいアプリケーションバージョンが作成されてリストに追加されます。

```
Select an application to use
1) [ Create new Application ]
(default is 1): 1
```

4. デフォルトのアプリケーション名は、eb init を実行するフォルダの名前です。プロジェクトを示す任意の名前を入力します。

```
Enter Application Name
(default is "eb"): eb
Application eb has been created.
```

5. ウェブアプリケーションが開発された言語またはフレームワークに一致するプラットフォームを選択します。アプリケーションの開発を始める以前の場合には、好みのプラットフォームを選択します。サンプルアプリケーションを起動する方法が簡略に確認でき、後でいつでもこの設定を変更できます。

```
Select a platform.
1) .NET Core on Linux
2) .NET on Windows Server
3) Docker
4) Go
5) Java
6) Node.js
7) PHP         <== select platform by number
8) Packer
9) Python
10) Ruby
11) Tomcat
(make a selection):7
```

6. 特定のプラットフォームブランチを選択します。

```
Select a platform branch.
1) PHP 8.4 running on 64bit Amazon Linux 2023
2) PHP 8.3 running on 64bit Amazon Linux 2023
3) PHP 8.2 running on 64bit Amazon Linux 2023
4) PHP 8.1 running on 64bit Amazon Linux 2023
5) PHP 8.1 running on 64bit Amazon Linux 2
(default is 1):1
```

7. [はい] を選択して、Elastic Beanstalk 環境のインスタンスに SSH キーペアを割り当てます。これにより、トラブルシューティングのために直接接続することができるようになります。

```
Do you want to set up SSH for your instances?
(y/n): y
```

8. 既存のキーペアを選択するか、新しいキーペアを作成します。eb init を使用して新しいキーペアを作成するには、ssh-keygen がローカルマシンにインストールされており、コマンドラインから呼び出せる必要があります。EB CLI は新しいキーペアを Amazon EC2 に登録して、プライベートキーをユーザーディレクトリの `.ssh` という名前のフォルダにローカルで保存します。

```
Select a keypair.
1) [ Create new KeyPair ]
(default is 1): 1
```

EB CLI のインストールは設定が完了し、使用可能となりました。

高度な設定

- .ebignore を使用してファイルを無視する
- 名前を指定されたプロファイルを使用する
- プロジェクトフォルダの代わりにアーティファクトをデプロイする
- 構成設定と優先順位
- インスタンスメタデータ

.ebignore を使用してファイルを無視する

`.ebignore` ファイルをディレクトリに追加して、プロジェクトディレクトリの特定のファイルを無視するように EB CLI に指示することができます。このファイルは、`.gitignore` ファイルと同様に

動作します。プロジェクトディレクトリを Elastic Beanstalk にデプロイし、新しいアプリケーションバージョンを作成するとき、EB CLI は、作成するソースバンドルに、`.ebignore` で指定されたファイルを含めません。

`.ebignore` が存在しないが `.gitignore` がある場合、EB CLI は `.gitignore` に指定されたファイルを無視します。`.ebignore` ファイルがある場合は、EB CLI は `.gitignore` を読み取りません。

`.ebignore` が存在する場合、EB CLI は git コマンドを使用せずにソースバンドルを作成します。つまり、EB CLI は `.ebignore` で指定されたファイルを無視し、他のすべてのファイルを含めます。特に、コミットされていないソースファイルが含まれます。

> **Note**
>
> Windows で `.ebignore` を追加すると、EB CLI がシンボリックリンクに従って、ソースバンドルの作成時にリンクファイルを含めます。これは既知の問題で、今後のアップデートで修正される予定です。

名前を指定されたプロファイルを使用する

名前を指定したプロファイルとして認証情報を `credentials` または `config` ファイルに保存する場合、`--profile` オプションを使用して明示的にプロファイルを指定できます。たとえば、次のコマンドでは `user2` プロファイルを使用して新しいアプリケーションを作成します。

```
$ eb init --profile user2
```

`AWS_EB_PROFILE` 環境変数を設定して、デフォルトのプロファイルの変更をすることもできます。この変数を設定すると、EB CLI は `default` や eb-cli の代わりに特定のプロファイルからの認証情報を読み込みます。

Linux、macOS、または Unix

```
$ export AWS_EB_PROFILE=user2
```

Windows

```
> set AWS_EB_PROFILE=user2
```

プロジェクトフォルダの代わりにアーティファクトをデプロイする

独立したビルドプロセスで作成した ZIP ファイルや WAR ファイルをデプロイするよう EB CLI に命令することができます。このためには、次の行をプロジェクトフォルダの `.elasticbeanstalk/config.yml` に追加します。

```
deploy:
  artifact: path/to/buildartifact.zip
```

Git リポジトリで EB CLI を設定して、ソースにアーティファクトをコミットしなかった場合は、`--staged` オプションを使用して最新のビルドをデプロイします。

```
~/eb$ eb deploy --staged
```

構成設定と優先順位

EB CLI はプロバイダーチェーンを使用して、システムまたはユーザー環境変数やローカル AWS 設定ファイルなど、さまざまな場所で AWS 認証情報を検索します。

EB CLI では、以下の順序で認証情報と構成設定が検索されます。

1. コマンドラインオプション – `--profile` を使用して名前付きプロファイルを指定し、デフォルトの設定を上書きします。
2. 環境変数 – `AWS_ACCESS_KEY_ID` および `AWS_SECRET_ACCESS_KEY`
3. AWS 認証情報ファイル – Linux および OS X システム`~/.aws/credentials`の場合は 、Windows システム`C:\Users\USERNAME\.aws\credentials`の場合は にあります。このファイルには、デフォルトのプロファイルに加えて、複数の名前付きプロファイルを含めることができます。
4. AWS CLI 設定ファイル – `~/.aws/config` (Linux および OS X システム) または `C:\Users\USERNAME\.aws\config` (Windows システム) にあります。このファイルには、デフォルトのプロファイル、名前付きプロファイル、およびそれぞれの AWS CLI固有の設定パラメータを含めることができます。
5. レガシー EB CLI 設定ファイル – `~/.elasticbeanstalk/config` (Linux および OS X システム) または `C:\Users\USERNAME\.elasticbeanstalk\config` (Windows システム) にあります。

6. インスタンスプロファイルの認証情報 - これらの認証情報は、インスタンスロールが割り当てられた Amazon EC2 インスタンスで使用でき、Amazon EC2 メタデータサービスを介して提供されます。インスタンスプロファイルには、Elastic Beanstalk を使用するアクセス許可が必要です。

認証情報ファイルに「eb-cli」という名前付きプロファイルが含まれている場合、EB CLI では、デフォルトのプロファイルよりもそのプロファイルが優先されます。プロファイルが見つからないか、プロファイルが見つかっても Elastic Beanstalk を使用するためのアクセス許可がない場合、EB CLI では、キーを入力するように求められます。

インスタンスメタデータ

Amazon EC2 インスタンスから EB CLI を使用するには、必要なリソースへのアクセス許可を持つロールを作成し、そのロールをインスタンスにその起動時に割り当てます。インスタンスを起動し、`pip` で EB CLI をインストールします。

```
~$ sudo pip install awsebcli
```

`pip` は Amazon Linux にプリインストールされています。

EB CLI はインスタンスのメタデータから認証情報を読み取ります。詳細については、IAM ユーザーガイドのAmazon EC2 インスタンスで実行されるアプリケーションに AWS リソースへのアクセス権を付与する」を参照してください。

Git での EB CLI の使用

EB CLI は Git 統合を提供するため、コードブランチを Elastic Beanstalk の特定の環境に関連付けてデプロイを整理できます。

Git をインストールして Git リポジトリを初期化するには

1. http://git-scm.com にアクセスして、Git の最新バージョンをダウンロードします。
2. 次のコマンドを入力して、Git リポジトリを初期化します。

   ```
   ~/eb$ git init
   ```

   これで、EB CLI は、アプリケーションが Git を使ってセットアップされることを認識します。
3. eb init をまだ実行していない場合は、ここで実行します。

```
~/eb$ eb init
```

Elastic Beanstalk 環境を Git ブランチに関連付ける

コードの各ブランチに異なる環境を関連付けることができます。ブランチをチェックアウトすると、変更は関連付けられた環境にデプロイされます。例えば、次のように入力すると、本番環境をメインラインブランチに関連付け、別の開発環境を開発ブランチに関連付けることができます。

```
~/eb$ git checkout mainline
~/eb$ eb use prod
~/eb$ git checkout develop
~/eb$ eb use dev
```

変更のデプロイ

デフォルトでは、EB CLI はコミット ID およびメッセージをそれぞれアプリケーションバージョンラベルと説明として使用して、最新のコミットを現在のブランチでデプロイします。コミットしない環境にデプロイする場合は、`--staged` を使用して、ステージング領域に追加された変更をデプロイするオプションを使用できます。

変更をコミットしないでデプロイするには

1. 新しいファイルと変更されたファイルをステージング領域に追加します。

   ```
   ~/eb$ git add .
   ```

2. eb deploy を使用してステージングされた変更をデプロイします。

   ```
   ~/eb$ eb deploy --staged
   ```

EB CLI がアーティファクトをデプロイするように設定しており、アーティファクトを Git リポジトリにコミットしていない場合は、`--staged` オプションを使用して最新のビルドをデプロイします。

Git サブモジュールの使用

一部のコードプロジェクトは、Git サブモジュール (最上位レポジトリ内のレポジトリ) から利点を得られます。eb create または eb deploy を使用してコードをデプロイするときに、EB CLI は、アプリケーションバージョンの zip ファイルにサブモジュールを含め、コードと他の部分とともにアップロードできます。

プロジェクトフォルダの EB CLI 設定ファイル `include_git_submodules` の `global` セクションで `.elasticbeanstalk/config.yml` オプションを使用して、サブモジュールのインクルードを制御できます。

サブモジュールを含めるには、このオプションを `true` に設定します。

```
global:
  include_git_submodules: true
```

`include_git_submodules` オプションが見つからないか、`false` に設定されている場合、EB CLI はアップロードされた zip ファイルにサブモジュールを含めません。

Git サブモジュールの詳細については、「Git ツール - サブモジュール」を参照してください。

> **デフォルトの動作**
>
> eb init を実行してプロジェクトを設定すると、EB CLI は `include_git_submodules` オプションを追加し、`true` に設定します。これにより、プロジェクト内のサブモジュールが確実にデプロイに含まれます。
> EB CLI は常にサブモジュールのインクルードをサポートしていませんでした。サブモジュールのサポートが追加される前に存在していた、プロジェクトに対する偶発的または望ましくない変更を避けるため、`include_git_submodules` オプションが見つからない場合、EB CLI はサブモジュールを含めません。これら既存のプロジェクトの 1 つがあり、デプロイにサブモジュールを含める場合は、このセクションで説明しているように、オプションを追加し、それを `true` に設定します。

> **CodeCommit の動作**
>
> Elastic Beanstalk の CodeCommit との統合は、現時点ではサブモジュールをサポートしていません。環境で CodeCommit との統合を有効にした場合、サブモジュールはデプロイに含まれません。

Git タグのアプリケーションバージョンへの割り当て

Git タグをバージョンラベルとして使用し、環境で実行されているアプリケーションバージョンを識別できます。たとえば、次のコマンドを入力します。

```
~/eb$ git tag -a v1.0 -m "My version 1.0"
```

EB CLI コマンドリファレンス

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用して、Elastic Beanstalk のアプリケーションおよび環境のデプロイと管理を行うためのさまざまなオペレーションを実行できます。Git によってソース管理されるアプリケーションのソースコードをデプロイする場合は、EB CLI が Git と連動します。詳細については、「Elastic Beanstalk を管理するための EB コマンドラインインターフェイス (EB CLI) の設定」および「Git での EB CLI の使用」を参照してください。

コマンド

- 一般的なオプション
- eb abort
- eb appversion
- eb clone
- eb codesource
- eb config
- eb console
- eb create
- eb deploy
- eb events
- eb health
- eb init
- eb labs
- eb list
- eb local
- eb logs
- eb migrate
- eb open
- eb platform
- eb printenv
- eb restore

- eb scale
- eb setenv
- eb ssh
- eb status
- eb swap
- eb tags
- eb terminate
- eb upgrade
- eb use

一般的なオプション

以下のオプションは、すべての EB CLI コマンドで使用できます。

名前	説明
`--debug`	デバッグの情報を出力します。
`-h, --help`	ヘルプメッセージを示します。 型: 文字列 デフォルト: なし
`--no-verify-ssl`	SSL 証明書認証をスキップします。プロキシで CLI を使用するのに問題がある場合、このオプションを使用します。
`--profile`	AWS 認証情報ファイルから特定のプロファイルを使用します。
`--quiet`	コマンドからのすべての出力を非表示にします。
`--region`	指定されたリージョンを使用します。
`-v, --verbose`	詳細な情報を表示します。

eb abort

説明

インスタンスへの環境設定の変更が進行中である場合に、アップグレードをキャンセルします。

> **Note**
> 更新が進行中の環境が 3 つ以上ある場合は、変更をロールバックする環境名を選択するよう求められます。

構文

eb abort

eb abort ***environment-name***

オプション

名前	説明
一般的なオプション	

Output

コマンドによって、現在更新中の環境の一覧が表示され、中止する更新を選択するよう求められます。現在更新中の環境が 1 つのみの場合、環境名を指定する必要はありません。成功すると、コマンドは環境設定の変更を戻します。ロールバックプロセスは、環境のすべてのインスタンスに以前の環境設定が戻るか、ロールバックプロセスが失敗するまで続行されます。

例

次の例では、プラットフォームのアップグレードをキャンセルします。

```
$ eb abort
Aborting update to environment "tmp-dev".
<list of events>
```

eb appversion

説明

EB CLI `appversion` コマンドは、Elastic Beanstalk アプリケーションのバージョンを管理します。デプロイせずにアプリケーションの新しいバージョンを作成したり、アプリケーションのバージョンを削除したり、アプリケーションバージョンのライフサイクルポリシーを作成したりできます。オプションを指定しないでコマンドを呼び出すと、インタラクティブモードになります。

`--create` オプションを使用して、アプリケーションの新しいバージョンを作成します。

`--delete` オプションを使用してアプリケーションのバージョンを削除します。

`lifecycle` オプションを使用してアプリケーションバージョンライフサイクルポリシーを表示または作成します。詳細については、「the section called “バージョンライフサイクル”」を参照してください。

構文

eb appversion

eb appversion [-c | --create]

eb appversion [-d | --delete] ***`version-label`***

eb appversion lifecycle [-p | --print]

オプション

名前	説明 型: 文字列
-a *`application-name`* または --application_name *`application-name`*	アプリケーションの名前。指定した名前のアプリケーションが見つからない場合、EB CLI によって新しいアプリケーションのアプリケーションバージョンが作成されます。 `--create` オプションでのみ適用できます。 型: 文字列

名前	説明 型: 文字列
-c または --create	アプリケーションの新しいバージョンを作成します。
-d *version-label* または --delete *version-label*	*version-label* というラベルの付いたアプリケーションのバージョンを削除します。
`-l` *`version_label`* または `--label` *`version_label`*	EB CLI で作成するバージョンに使用するラベルを指定します。このオプションを使用しない場合、EB CLI は新しい一意のラベルを生成します。バージョンラベルを指定する場合は、そのラベルが一意であることを確認してください。 `--create` オプションでのみ適用できます。 型: 文字列
ライフサイクル	新しいアプリケーションバージョンライフサイクルポリシーを作成するには、デフォルトのテキストエディタを呼び出します。このポリシーを使用して、アプリケーションバージョンクォータに達しないようにします。
lifecycle -p または lifecycle --print	現行のアプリケーションライフサイクルポリシーを表示します。

名前	説明 型: 文字列
-m "*version_description* " または --message "*version_description* "	アプリケーションバージョンの説明。二重引用符で囲まれています。 --create オプションでのみ適用できます。 型: 文字列
-p または --process	ソースバンドル内の環境マニフェストおよび設定ファイルを事前処理し、検証します。設定ファイルを検証すると、問題を特定できます。アプリケーションバージョンを環境にデプロイする前に、これを実行することをお勧めします。 --create オプションでのみ適用できます。
--source codecommit/ *repository-name*/*branch-name*	CodeCommit リポジトリとブランチ。 --create オプションでのみ適用できます。
--staged	HEAD コミットではなく git インデックスでステージングされたファイルを使用して、アプリケーションバージョンを作成します。 --create オプションでのみ適用できます。
--timeout *#*	コマンドがタイムアウトするまでの時間 (分) 。 --create オプションでのみ適用できます。
一般的なオプション	

コマンドのインタラクティブな使用

引数なしでコマンドを使用すると、出力にはアプリケーションのバージョンが表示されます。それらは時系列の逆順で一覧表示され、最新のバージョンが最初に表示されています。画面の外観につい

ては、[例] セクションを参照してください。ステータス行が下部に表示されることに注意してください。ステータス行には、状況依存情報が表示されます。

d を押すとアプリケーションバージョンが表示され、l を押すとアプリケーションのライフサイクルポリシーを管理でき、q を押すと変更を加えずに終了できます。

> **Note**
>
> バージョンが環境にデプロイされた場合、そのバージョンを削除することはできません。

Output

`--create` オプションを指定してコマンドを実行すると、アプリケーションバージョンが作成されたことを確認するメッセージが表示されます。

`--delete` *`version-label`* オプションを指定してコマンドを実行すると、アプリケーションバージョンが削除されたことを確認するメッセージが表示されます。

例

次の例では、デプロイのないアプリケーション用のインタラクティブなウィンドウを示します。

次の例では、アプリケーションのインタラクティブウィンドウが、4 つ目のバージョン (バージョンラベルは [Sample Application (サンプルアプリケーション)]) がデプロイされて表示されます。

次の例は、eb appversion lifecycle -p コマンドの出力を示します。*`ACCOUNT-ID`* はユーザーのアカウント ID です。

```
Application details for: lifecycle
  Region: sa-east-1
  Description: Application created from the EB CLI using "eb init"
  Date Created: 2016/12/20 02:48 UTC
  Date Updated: 2016/12/20 02:48 UTC
  Application Versions: ['Sample Application']
  Resource Lifecycle Config(s):
    VersionLifecycleConfig:
      MaxCountRule:
```

```
        DeleteSourceFromS3: False
        Enabled: False
        MaxCount: 200
      MaxAgeRule:
        DeleteSourceFromS3: False
        Enabled: False
        MaxAgeInDays: 180
    ServiceRole: arn:aws:iam::ACCOUNT-ID:role/aws-elasticbeanstalk-service-role
```

eb clone

説明

環境のクローンを新しい環境として作成します。元の環境とクローンされた環境では設定が同じになります。

> **Note**
>
> デフォルトでは、クローンを作成する環境のソリューションスタックのバージョンに関係なく、eb clone コマンドは最新のソリューションスタックでクローン環境を作成します。コマンドの実行時に `--exact` オプションを含めることで、これを抑制できます。

> **Important**
>
> クローンの Elastic Beanstalk 環境は、イングレス用のセキュリティグループを引き継ぐのではなく、環境はすべてのインターネットトラフィックに対してオープンのままになります。クローンの環境のイングレスセキュリティグループを再確立する必要があります。
> 環境設定のドリフトステータスを確認することで、クローン化できないリソースを確認できます。詳細については、「AWS CloudFormation ユーザーガイド」の「CloudFormation スタック全体のドリフトを検出する」を参照してください 。

構文

eb clone

eb clone ***environment-name***

オプション

名前	説明
-n *string* または --clone_name *string*	クローンの環境に指定する名前。
-c *string* または --cname *string*	クローンの環境で必要となる CNAME プレフィックス。
--envvars	環境プロパティ。「*name*=*value*」の形式を使用してカンマ区切りリストで指定します。 型: 文字列 制約: • キーと値のペアはカンマで区切る必要があります。 • キーと値には、任意の言語のアルファベット文字、数字、空白、表示されない区切り文字、記号 (_ . : / + \ - @) を使用できます。 • キーの長さは最大 128 文字です。値の長さは最大 256 文字です。 • キーと値は大文字と小文字が区別されます。 • 値と環境名を同じすることはできません。 • 値に aws: または elasticbeanstalk: を含めることはできません。 • すべての環境プロパティの合計サイズが 4,096 バイトを超えることはできません。

名前	説明
`--exact`	Elastic Beanstalk が新しいクローン環境のソリューションスタックのバージョンを使用可能な最新のバージョンに更新することを防止します (元の環境プラットフォーム用)。
`--scale` *number*	クローンの環境を起動したときにその環境で実行されるインスタンスの数。
`--tags` *name*=*value*	お客様の環境内のリソースのタグ。*name*=*value* の形式を使用してカンマ区切りリストで指定します。
`--timeout`	コマンドがタイムアウトするまでの時間 (分) 。
一般的なオプション	

Output

成功すると、コマンドは元の環境と同じ設定を含む環境、または eb clone のオプションで指定された変更が適用された環境を作成します。

例

次の例では、指定した環境のクローンが作成されます。

```
$ eb clone
Enter name for Environment Clone
(default is tmp-dev-clone):
Enter DNS CNAME prefix
(default is tmp-dev-clone):
Environment details for: tmp-dev-clone
  Application name: tmp
  Region: us-west-2
  Deployed Version: app-141029_144740
  Environment ID: e-vjvrqnn5pv
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev-clone.elasticbeanstalk.com
  Updated: 2014-10-29 22:00:23.008000+00:00
Printing Status:
```

```
2018-07-11 21:04:20    INFO: createEnvironment is starting.
2018-07-11 21:04:21    INFO: Using elasticbeanstalk-us-west-2-888888888888 as Amazon S3
 storage bucket for environment data.
...
2018-07-11 21:07:10    INFO: Successfully launched environment: tmp-dev-clone
```

eb codesource

説明

EB CLI を設定して CodeCommit リポジトリからデプロイするか、または CodeCommit 統合を無効にし、ローカルマシンからソースバンドルをアップロードします。

> **Note**
> 一部の AWS リージョンでは CodeCommit が提供されていません。それらのリージョンでは、Elastic Beanstalk と CodeCommit の統合は有効ではありません。
> 各リージョンで提供される AWS サービスの詳細については、「リージョンテーブル」を参照してください。

構文

eb codesource

eb codesource codecommit

eb codesource local

オプション

名前	説明
一般的なオプション	

Output

eb codesource は、CodeCommit 統合とスタンダードデプロイのどちらかを選択するよう求めます。

eb codesource codecommit は、CodeCommit 統合のインタラクティブなリポジトリ設定を開始します。

eb codesource local は元の設定を示し、CodeCommit 統合を無効にします。

例

現在のブランチの CodeCommit 統合を設定する場合は、eb codesource codecommit を使用します。

```
~/my-app$ eb codesource codecommit
Select a repository
1) my-repo
2) my-app
3) [ Create new Repository ]
(default is 1): 1

Select a branch
1) mainline
2) test
3) [ Create new Branch with local HEAD ]
(default is 1): 1
```

現在のブランチの CodeCommit 統合を無効にする場合は、eb codesource local を使用します。

```
~/my-app$ eb codesource local
Current CodeCommit setup:
  Repository: my-app
  Branch: mainline
Default set to use local sources
```

eb config

説明

環境に適用中の設定と保存されている環境設定を管理します。このコマンドを使用して、保存されている環境設定をアップロード、ダウンロード、または一覧表示できます。また、適用中の設定のダウンロード、表示、または更新にも使用できます。

root ディレクトリにカスタムプラットフォームを指定する `platform.yaml` ファイルが含まれている場合、このコマンドは ビルダー設定も変更します。これは、`platform.yaml` で設定された値に基づいて行われます。

> **Note**
>
> eb config は、環境プロパティを表示しません。アプリケーションで読み取り可能な環境プロパティを設定するには、代わりに eb setenv を使用します。

構文

eb config コマンドが環境に適用中の設定を操作するために使用する構文の一部を次に示します。例については、このトピックで後述する「例」セクションを参照してください。

- eb config – EDITOR 環境変数として設定したテキストエディタで、環境に適用中の設定を表示します。変更をファイルに保存してエディタを閉じると、ファイルに保存したオプション設定によって環境が更新されます。

  > **Note**
  >
  > EDITOR 環境変数を設定しなかった場合、EB CLI は YAML ファイルのデフォルトエディタにオプション設定を表示します。

- eb config ***environment-name*** – 名前付き環境の設定を表示および更新します。設定は、設定したテキストエディタまたはデフォルトのエディタの YAML ファイルに表示されます。
- eb config save – 現在の環境に適用中の設定を `.elasticbeanstalk/saved_configs/` というファイル名で `[configuration-name].cfg.yml` に保存します。デフォルトでは、EB CLI は設定を環境名に基いて *`configuration-name`* という名前で保存します。コマンドの実行時に `--cfg` オプションと必要な設定名を含めることで、別の設定名を指定できます。

  `--tags` オプションを使用して保存された設定にタグを付けることができます。
- eb config **`--display`** – 環境に適用中の設定をファイルではなく stdout に書き込みます。デフォルトでは、これによって端末に設定が表示されます。
- eb config **`--update`** ***`configuration_string`*** **`|`** ***`file_path`*** – 現在の環境に適用されている設定を、*`configuration_string`* または *`file_path`* で識別されるファイル内で指定された情報で更新します。

> **Note**
>
> `--display` オプションおよび `--update` オプションは、環境の設定をプログラム的に読み込んだり改訂したりする柔軟性を備えています。

eb config コマンドを使用して保存した設定を操作するための構文について次に説明します。例については、このトピックで後述する「例」セクションを参照してください。

- eb config get ***config-name*** – Amazon S3 から名前付きで保存された設定をダウンロードします。
- eb config delete ***config-name*** – Amazon S3 から名前付きで保存された設定を削除します。また、すでにダウンロードしている場合は、ローカルで削除します。
- eb config list – Amazon S3 内に保存した設定を一覧表示します。
- eb config put ***filename*** – Amazon S3 バケットに名前を付けて保存した設定をアップロードします。*filename* にはファイル拡張子 `.cfg.yml` が必要です。パスを指定せずにファイル名を指定するには、`.elasticbeanstalk` フォルダまたは `.elasticbeanstalk/saved_configs/` フォルダにファイルを保存してからコマンドを実行します。また、「*filename*」にフルパスを指定できます。

オプション

名前	説明
`--cfg` *config-name*	保存した設定に使用する名前。 このオプションは、eb config save でのみ使用できます。

`-d`
 	
または
 	
`--display`
	現在の環境の設定を表示します (stdout への書き込み)。
	
	`--format` オプションを使用して、出力が JSON か YAML かを指定します。指定しない場合、出力は YAML 形式になります。
	
	このオプションは、他のサブコマンドなしで eb config コマンドを使用する場合にのみ機能します。

`-f` *format_type*	表示形式を指定します。有効な値は JSON または YAML です。

名前	説明
または `--format format_type`	デフォルトは YAML です。 このオプションは、`--display` オプションでのみ機能します。
`--tags key1=value1[,ke`	保存された設定に追加するタグ。リストでタグを指定するときは、「キー=値」のペアとして指定し、それぞれをカンマで区切ります。 詳細については、「」を参照してください保存された設定にタグ付けする このオプションは、eb config save でのみ使用できます。
`--timeout timeout`	コマンドがタイムアウトするまでの時間（分）。

名前	説明

`-u` *`configuration_string`* | *`file_path`*
 	
または
 	
`--update` *`configuration_string`* | *`file_path`*
	現在の環境に適用中の設定を更新します。
	
	このオプションは、他のサブコマンドなしで eb config コマンドを使用する場合にのみ機能します。
	
	`configuration_string` | *`file_path`* パラメータは文字列型です。文字列で、環境の設定に対して追加、更新、または削除する名前空間と対応するオプションのリストを指定します。または、入力文字列で、これと同じ情報を含むファイルを表すこともできます。
	
	ファイル名を指定するには、入力文字列が `"file://<`*`path`*`><`*`filename`*`>"` の形式になっている必要があります。*`path`* なしでファイル名を指定するには、コマンドを実行するフォルダにファイルを保存します。また、フルパスでファイル名を指定することもできます。
	
	設定情報は、次の条件を満たす必要があります。少なくとも 、OptionSettings または OptionsToRemove のどちらか 1 つのセクションが必要です。OptionSettings を使用して、オプションを追加または変更します。OptionsToRemove を使用して、名前空間からオプションを削除します。例については、このトピックで後述する「例」セクションを参照してください。
	
	Example
	
	YAML 形式
	
	`OptionSettings:`
	`  namespace1:`
	`  option-name-1:` *`option-value-1`*
	`  option-name-2:` *`option-value-2`*
	`  ...`
	`OptionsToRemove:`
	`  namespace1:`
	`  option-name-1`
	`  option-name-2`
	`  ...`

名前	説明
	Example JSON 形式 `{ "OptionSettings": { "namespace1": { "option-name-1": " option-value-1 ", "option-name-2": " option-value-2 ", ... } }, "OptionsToRemove": { "namespace1": { "option-name-1", "option-name-2", ... } } }`
一般的なオプション	

Output

サブコマンドやオプションを追加していない状態で eb config または eb config ***environment-name*** コマンドが正常に実行されると、EDITOR 環境変数として設定したテキストエディタに現在のオプション設定が表示されます。EDITOR 環境変数を設定しなかった場合、EB CLI は YAML ファイルのデフォルトエディタにオプション設定を表示します。

変更をファイルに保存してエディタを閉じると、ファイルに保存したオプション設定によって環境が更新されます。次の出力は、設定の更新を確認するために表示されます。

```
$ eb config myApp-dev
    Printing Status:
    2021-05-19 18:09:45    INFO    Environment update is starting.
    2021-05-19 18:09:55    INFO    Updating environment myApp-dev's configuration
 settings.
```

```
    2021-05-19 18:11:20    INFO    Successfully deployed new configuration to
 environment.
```

`--display` オプションを指定してコマンドが正常に実行されると、現在の環境の設定が表示されます (stdout への書き込み)。

`get` パラメータを指定したコマンドの実行に成功すると、コマンドはダウンロードしたローカルコピーの場所を表示します。

`save` パラメータを指定したコマンドの実行に成功すると、コマンドは保存したファイルの場所を表示します。

例

このセクションでは、オプション設定ファイルの表示や編集に使用するテキストエディタを変更する方法を説明します。

Linux および UNIX の場合、次の例では、エディタが vim に変更されます。

```
$ export EDITOR=vim
```

Linux および UNIX の場合、次の例では、エディタが `/usr/bin/kate` にインストールされているエディタに変更されます。

```
$ export EDITOR=/usr/bin/kate
```

次の例では、エディタが Notepad++ に変更されます (Windows の場合) 。

```
> set EDITOR="C:\Program Files\Notepad++\Notepad++.exe
```

このセクションでは、サブコマンドとともに eb config コマンドを実行する場合の例を示します。

以下の例は `app-tmp` という名前の保存された設定を削除します。

```
$ eb config delete app-tmp
```

以下の例は Amazon S3 バケットから app-tmp という名前の保存済み設定をダウンロードします。

```
$ eb config get app-tmp
```

以下の例は Amazon S3 バケットに保存されている保存済み設定の名前を一覧表示します。

```
$ eb config list
```

以下の例は Amazon S3 バケットに app-tmp という名前の保存済み設定のローカルコピーをアップロードします。

```
$ eb config put app-tmp
```

以下の例は現在実行中の環境から設定を保存します。保存した設定に使用する名前を指定しない場合、Elastic Beanstalk は環境名に従って設定ファイルに名前を付けます。例えば、tmp-dev という名前の環境は `tmp-dev.cfg.yml` と呼ばれます。Elastic Beanstalk はファイルをフォルダ `/.elasticbeanstalk/saved_configs/` に保存します。

```
$ eb config save
```

以下の例では、`--cfg` オプションを使用して環境 tmp-dev から `v1-app-tmp.cfg.yml` というファイルに設定を保存しています。Elastic Beanstalk はファイルをフォルダ `/.elasticbeanstalk/saved_configs/` に保存します。環境名を指定しない場合、Elastic Beanstalk は現在実行中の環境から設定を保存します。

```
$ eb config save tmp-dev --cfg v1-app-tmp
```

このセクションでは、サブコマンドを使用せずに eb config コマンドを実行する場合の例を示します。

次のコマンドは、現在の環境のオプション設定をテキストエディタに表示します。

```
$ eb config
```

次のコマンドは、my-env 環境のオプション設定をテキストエディタに表示します。

```
$ eb config my-env
```

次の例では、現在の環境のオプション設定を表示します。`--format` オプションに特定の形式が指定されていないため、YAML 形式で出力されます。

```
$ eb config --display
```

次の例では、現在の環境のオプション設定を、`example.txt` という名前のファイルに指定されているように更新します。ファイルは YAML 形式または JSON 形式のいずれかです。EB CLI はファイル形式を自動的に検出します。

- 名前空間 `aws:autoscaling:asg` の Minsize オプションは 1 に設定されます。
- 名前空間 `aws:elasticbeanstalk:command` のバッチサイズは 30% に設定されます。
- これにより、名前空間 `AWSEBV2LoadBalancer.aws:elbv2:loadbalancer` から IdleTimeout: None のオプション設定が削除されます。

```
$ eb config --update "file://example.txt"
```

Example - ファイル名: **`example.txt`** - YAML 形式

```
OptionSettings:
  'aws:elasticbeanstalk:command':
    BatchSize: '30'
    BatchSizeType: Percentage
  'aws:autoscaling:asg':
    MinSize: '1'
OptionsToRemove:
  'AWSEBV2LoadBalancer.aws:elbv2:loadbalancer':
    IdleTimeout
```

Example - ファイル名: **`example.txt`** - JSON 形式

```
{
    "OptionSettings": {
        "aws:elasticbeanstalk:command": {
            "BatchSize": "30",
            "BatchSizeType": "Percentage"
        },
        "aws:autoscaling:asg": {
            "MinSize": "1"
        }
    },
    "OptionsToRemove": {
        "AWSEBV2LoadBalancer.aws:elbv2:loadbalancer": {
            "IdleTimeout"
        }
    }
```

```
}
```

次の例では、現在の環境のオプション設定を更新します。このコマンドは、`aws:autoscaling:asg` 名前空間の Minsize オプションを 1 に設定します。

> **Note**
>
> これらの例は、Windows PowerShell に固有のものです。二重引用符 (") 文字の前にスラッシュ (\) 文字を付けてエスケープし、二重引用符という文字そのものであることを示します。オペレーティングシステムやコマンドライン環境が異なると、エスケープシーケンスが異なる場合があります。このため、前の例に示した file オプションを使用することをお勧めします。ファイルに設定オプションを指定する場合は、エスケープ文字を必要とせず、異なるオペレーティングシステム間で一貫性があります。

次の例は JSON 形式です。EB CLI は、形式が JSON または YAML のどちらであるかを検出します。

```
PS C:\Users\myUser\EB_apps\myApp-env>eb config --update '{\"OptionSettings\":
{\"aws:autoscaling:asg\":{\"MaxSize\":\"1\"}}}'
```

次の例は YAML 形式です。コマンドに YAML ファイルで必要なスペースと行末のリターンが含まれ、YAML 文字列が正しい形式で入力されています。

- 各行は「Enter」または「Return」キーで終わります。
- 2 行目を 2 つのスペースで開始し、3 行目を 4 つのスペースで開始します。

```
PS C:\Users\myUser\EB_apps\myApp-env>eb config --update 'OptionSettings:
>>  aws:autoscaling:asg:
>>    MinSize: \"1\"'
```

eb console

説明

ブラウザを開き、環境設定ダッシュボードを Elastic Beanstalk マネジメントコンソールで表示します。

root ディレクトリにカスタムプラットフォームを指定する `platform.yaml` ファイルが含まれている場合、このコマンドは Elastic Beanstalk 管理コンソールで `platform.yaml` で指定されているビルダー環境設定も表示します。

構文

eb console

eb console ***environment-name***

オプション

名前	説明
一般的なオプション	

eb create

説明

新しい環境を作成し、アプリケーションバージョンをデプロイします。

> **Note**
>
> - .NET アプリケーションで eb create を使用するには、「.NET アプリケーションのソースバンドルの作成」を参照してデプロイパッケージを作成し、「プロジェクトフォルダの代わりにアーティファクトをデプロイする」を参照してパッケージをアーティファクトとしてデプロイするように CLI 設定をセットアップする必要があります。
> - EB CLI を使用して環境を作成するには、サービスロールが必要です。Elastic Beanstalk コンソールで環境を作成することで、サービスロールを作成できます。サービスロールがない場合は、`eb create` を実行すると EB CLI で作成されます。

アプリケーションバージョンは、次のいくつかのソースからデプロイできます。

- デフォルト: ローカルプロジェクトディレクトリのアプリケーションソースコード。
- `--version` オプションを使用: アプリケーションに既に存在するアプリケーションバージョン。

- プロジェクトディレクトリにアプリケーションコードが含まれていない場合、または `--sample` オプションを使用している場合: ご使用環境のプラットフォーム固有のサンプルアプリケーション。

構文

eb create

eb create ***environment-name***

環境名は 4 ~ 40 文字の長さにする必要があります。名前には、英字、数字、ハイフン (-) のみを使用できます。環境名の先頭および末尾にはハイフンを使用できません。

コマンドに環境名を含める場合、EB CLI は選択やサービスロールの作成を要求しません。

環境名引数を指定せずにコマンドを実行すると、インタラクティブなフローで実行され、一部の設定で値の入力または選択が求められます。このインタラクティブなフローで、サンプルアプリケーションをデプロイすると、EB CLI は、このサンプルアプリケーションをローカルプロジェクトディレクトにダウンロードするかどうかを確認します。ダウンロードすると、後で新しい環境で EB CLI を使用して、アプリケーションのコードを必要とするオペレーション (例: eb deploy) を実行できます。

一部のインタラクティブなフロープロンプトは、特定の条件でのみ表示されます。例えば、アプリケーションロードバランサーの使用を選択し、アカウントに共有可能な Application Load Balancer が少なくとも 1 つある場合、Elastic Beanstalk は共有ロードバランサーを使用するかどうかを尋ねるプロンプトを表示します。共有可能な Application Load Balancer がアカウントにない場合、このプロンプトは表示されません。

オプション

これらのオプションはいずれも必須ではありません。オプションを指定せずに eb create を実行すると、EB CLI より、各設定の値を入力または選択するよう求められます。

名前	説明

`-d`
または
`--branch_default`
	現在のリポジトリのデフォルト環境として環境を設定します。

名前	説明
`--cfg` *`config-name`*	`.elasticbeanstalk/saved_configs/` または Amazon S3 バケットに保存されている設定からプラットフォームの設定を使用します。`.cfg.yml` 拡張子なしで、ファイル名のみを指定します。
`-c` *`subdomain-name`* または `--cname` *`subdomain-name`*	ウェブサイトにルーティングする CNAME DNS エントリの前に付けるサブドメイン名。 型: 文字列 デフォルト: 環境の名前

`-db`
 	
または
 	
`--database`
	データベースを環境にアタッチします。eb create オプションを指定して `--database` を実行するときに、`--database.username` オプションと `--database.password` オプションを指定しないと、EB CLI では、データベースのマスターユーザー名とパスワードを指定するように求められます。

`-db.engine` *`engine`* または `--database.engine` *`engine`*	データベース エンジンのタイプ。このオプションを指定して eb create を実行すると、EB CLI はデータベースをアタッチして環境を起動します。これは、`--database` オプションを指定してコマンドを実行しなかった場合でも同様です。 型: 文字列 有効な値: `mysql`, `oracle-se1` , `postgres`, `sqlserver-ex` , `sqlserver-web` , `sqlserver-se`

名前	説明
`-db.i` *`instance_type`* または `--database.instance` *`instance_type`*	データベースに使用する Amazon EC2 インスタンスのタイプ。このオプションを指定して eb create を実行すると、EB CLI はデータベースをアタッチして環境を起動します。これは、`--database` オプションを指定してコマンドを実行しなかった場合でも同様です。 タイプ: 文字列 有効な値: Amazon RDS は DB インスタンスのスタンダードセットをsupport しています。DB エンジンに適切な DB インスタンスを選択するには、いくつかの特定の考慮事項をアカウントに取り入れる必要があります。詳細については、Amazon RDS ユーザーガイドの「DB インスタンス classes」を参照してください。
`-db.pass` *`password`* または `--database.password` *`password`*	データベースのパスワード。このオプションを指定して eb create を実行すると、EB CLI はデータベースをアタッチして環境を起動します。これは、`--database` オプションを指定してコマンドを実行しなかった場合でも同様です。

名前	説明
`-db.size` *number_of_gigabytes* または `--database.size` *number_of_gigabytes*	データベースストレージに割り当てるサイズ（ギガバイト（GB）単位の数値）。このオプションを指定して eb create を実行すると、EB CLI はデータベースをアタッチして環境を起動します。これは、`--database` オプションを指定してコマンドを実行しなかった場合でも同様です。 型: 数値 有効な値: • MySQL – 5 ~ 1024。デフォルト: 5。 • Postgres – 5 ~ 1024。デフォルト: 5。 • Oracle – 10 ~ 1024。デフォルト: 10。 • Microsoft SQL Server Express Edition – 30。 • Microsoft SQL Server Web Edition – 30。 • Microsoft SQL Server Standard Edition – 200。
`-db.user` *username* または `--database.username` *username*	データベースのユーザー名。このオプションを指定して eb create を実行すると、`--database` オプションを指定してコマンドを実行しなかった場合でも、EB CLI はデータベースをアタッチして環境を起動します。eb create オプションを指定して `--database` を実行するときに、`--database.username` オプションと `--database.password` オプションを指定しないと、EB CLI では、マスターデータベースのユーザー名とパスワードを指定するように求められます。
`-db.version` *version* または `--database.version` *#####*	データベースエンジンのバージョンの指定に使用します。このフラグがある場合、環境は `--database` フラグがなくても、指定したバージョン番号のデータベースを起動します。

名前	説明
`--elb-type` *`type`*	ロードバランサータイプ。 型: 文字列 有効な値: `classic`, `application` , `network` デフォルト: `application`

`-es`
 	
または
 	
`--enable-spot`
	環境のスポットインスタンスリクエストを有効にします。詳細については、「」を参照してくださいAuto Scaling グループ
	
	関連オプション:
	
	• `--instance-types`
	• `--on-demand-base-capacity`
	• `--on-demand-above-base-capacity`
	• `--spot-max-price`

`--env-group-suffix` *`groupname`*	環境の名前に連結するグループ名。環境を構成するのみで使用します。
`--envvars`	環境プロパティ。「*`name`*=*`value`*」の形式を使用してカンマ区切りリストで指定します。制限については、「環境プロパティ (環境変数) の設定」を参照してください。
`-ip` *`profile_name`* または `--instance_profile` *`profile_name`*	アプリケーションが AWS リソースにアクセスするために必要な一時的なセキュリティ認証情報を持つ IAM ロールを持つインスタンスプロファイル。

名前	説明

`-it`
 	
or
 	
`--instance-types` *`type1`*`[,`*`type2`*` ...]`
	環境で使用する Amazon EC2 インスタンスタイプのコンマ区切りリスト。このオプションを指定しない場合、Elastic Beanstalk はデフォルトのインスタンスタイプを提供します。
	
	詳細については、「Amazon EC2 インスタンス」および「Auto Scaling グループ」を参照してください。
	
	Important
	EB CLI は、このオプションをスポットインスタンスにのみ適用します。`--enable-spot` オプションは、このオプションとともに使用されない限り、EB CLI によって無視されます。オンデマンドインスタンスのインスタンスタイプを指定するには、代わりに `--intance-type` (「s」なし) オプションを使用します。

`-i`
 	
または
 	
`--instance_type`
	環境で使用する Amazon EC2 インスタンスタイプ。このオプションを指定しない場合、Elastic Beanstalk はデフォルトのインスタンスタイプを提供します。
	
	詳細については、「」を参照してくださいAmazon EC2 インスタンス
	
	Important
	EB CLI は、このオプションをオンデマンドインスタンスにのみ適用します。EB CLI によって無視されるため、このオプションとともに `--enable-spot` オプションを使用しないでください。スポットインスタンスのインスタンスタイプを指定するには、代わりに `--intance-types` (「s」あり) オプションを使用します。

名前	説明
`-k` *`key_name`* または `--keyname` *`key_name`*	Elastic Beanstalk アプリケーションを実行する Amazon EC2 インスタンスに安全にログインするために、Secure Shell (SSH) クライアントで使用する Amazon EC2 キーペアの名前。eb create コマンドでこのオプションを使用すると、指定した値は、eb init で指定したキー名を上書きします。 有効な値: Amazon EC2 に登録された既存のキー名
`-im` *`number-of-instances`* または `--min-instances` *`number-of-instances`*	環境に必要な Amazon EC2 インスタンスの最小数。 型: 数値 (整数) デフォルト: `1` 有効な値: `1 ~ 10000`
`-ix` *`number-of-instances`* または `--max-instances` *`number-of-instances`*	環境に許可する Amazon EC2 インスタンスの最大数。 型: 数値 (整数) デフォルト: `4` 有効な値: `1 ~ 10000`
`--modules` *`component-a component-b`*	作成する環境のコンポーネントリスト。ComposeEnvironments でのみ使用します。

名前	説明
-sb または --on-demand-base-capacity	環境のスケールアップ時にスポットインスタンスを考慮する前に、Auto Scaling グループがプロビジョニングするオンデマンドインスタンスの最小数。 このオプションは、--enable-spot オプションでのみ指定できます。詳細については、「」を参照してくださいAuto Scaling グループ 型: 数値 (整数) デフォルト: 0 有効な値: 0 ~ --max-instances (存在しない場合: MaxSize 名前空間の aws:autoscaling:asg オプション)
-sp または --on-demand-above-base-capacity	--on-demand-base-capacity オプションで指定されたインスタンス数を超えて Auto Scaling グループがプロビジョニングする追加容量の一部としてのオンデマンドインスタンスの割合。 このオプションは、--enable-spot オプションでのみ指定できます。詳細については、「Auto Scaling グループ」を参照してください。 型: 数値 (整数) デフォルト: 単一インスタンス環境では 0、負荷分散された環境では 70 有効な値: 0 ~ 100

名前	説明
`-p` *`platform-version`* または `--platform` *`platform-version`*	使用するプラットフォームバージョン。プラットフォーム、プラットフォームとバージョン、プラットフォームブランチ、ソリューションスタック名、またはソリューションスタック ARN を指定できます。次に例を示します。 • `php`、`PHP`、`node.js` – 指定されたプラットフォームの最新のプラットフォームバージョン • `php-7.2`、`"PHP 7.2"` – 推奨される (通常は最新の) PHP 7.2 プラットフォームバージョン • `"PHP 7.2 running on 64bit Amazon Linux"` – このプラットフォームブランチに推奨される (通常は最新の) PHP プラットフォームバージョン • `"64bit Amazon Linux 2017.09 v2.6.3 running PHP 7.1"` – ソリューションスタック名で指定された PHP プラットフォームバージョン • `"arn:aws:elasticbeanstalk:us-east-2::platform/PHP 7.1 running on 64bit Amazon Linux/2.6.3"` – ソリューションスタック ARN で指定された PHP プラットフォームバージョン 利用可能な設定のリストを取得するには、`eb platform list` を使用します。 `--platform` を指定した場合は、`eb init` 中に提供された値よりも優先されます。

`-pr`
 	
または
 	
`--process`
	ソースバンドル内の環境マニフェストおよび設定ファイルを事前処理し、検証します。設定ファイルを検証することで、アプリケーションバージョンを環境にデプロイする前に問題を特定できます。

名前	説明
`-r` *`region`* または `--region` *`region`*	アプリケーションをデプロイする AWS リージョン。 このオプションに指定できる値のリストについては、「AWS 全般のリファレンス」の「AWS Elastic Beanstalk エンドポイントとクォータ」を参照してください。
`--sample`	リポジトリのコードではなく、新しい環境にサンプルアプリケーションをデプロイします。
`--scale` *`number-of-instances`*	指定された数のインスタンスで起動します
`--service-role` *`servicerole`*	デフォルト以外のサービスロールを環境に割り当てます。 **Note** ARN を入力しないでください。ロール名のみを入力します。Elastic Beanstalk は、結果の ARN を内部的に作成するために、ロール名の先頭に正しい値を付けます。

名前	説明
`-ls` *`load-balancer`* または `--shared-lb` *`load-balancer`*	共有されたロードバランサーを使用する環境を設定します。アカウント内の共有可能なロードバランサーの名前または ARN を指定します。これは、別の Elastic Beanstalk 環境によって作成されたものではなく、お客様が明示的に作成した Application Load Balancer です。詳細については、「」を参照してください共有 Application Load Balancer パラメータの例: • `FrontEndLB` – ロードバランサー名。 • `arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/app/FrontEndLB/0dbf78d8ad96abbc` - Application Load Balancer の ARN。 このオプションは、`--elb-type application` でのみ指定できます。このオプションを指定し、`--shared-lb` を指定しない場合、Elastic Beanstalk は環境専用のロードバランサーを作成します。
`-lp` *`port`* または `--shared-lb-port` *`port`*	この環境の共有ロードバランサーのデフォルトのリスナーポート。Elastic Beanstalk は、このリスナーからのすべてのトラフィックをデフォルトの環境プロセスにルーティングするリスナールールを追加します。詳細については、「共有 Application Load Balancer」を参照してください。 型: 数値 (整数) デフォルト: `80` 有効な値: 共有ロードバランサーのリスナーポートを表す任意の整数。

名前	説明
`--single`	単一の Amazon EC2 インスタンスを使用して環境を作成します。ロードバランサーは使用しません。 **Warning** シングルインスタンス環境は、本稼働環境では利用できません。インスタンスがデプロイ中に不安定になった場合、または設定の更新中に Elastic Beanstalk がインスタンスを終了して再起動した場合は、アプリケーションを一定期間使用できなくなることがあります。開発、テスト、またはステージングには、単一インスタンス環境を使用します。本稼働用には負荷分散された環境を使用します。

`-sm`
 	
または
 	
`--spot-max-price`
	お客様がスポットインスタンスに対して支払ってもよいと考えるユニット時間あたりの上限価格 (米ドル)。
	
	このオプションは、`--enable-spot` オプションでのみ指定できます。詳細については、「Auto Scaling グループ」を参照してください。
	
	タイプ: 数値 (浮動小数点)
	
	デフォルト: 各インスタンスタイプのオンデマンド料金。この場合のオプションの値は `null` です。
	
	有効な値: `0.001` ~ `20.0`
	
	スポットインスタンスの上限価格オプションに関する推奨事項については、「Amazon EC2 ユーザーガイド」の「スポットインスタンスの料金履歴」を参照してください。

名前	説明
`--tags` *key1=value1[,key2=va*	環境内のリソースにタグ付けします。タグは、`key=value` ペアのカンマ区切りリストとして指定されます。 詳細については、「環境のタグ付け」を参照してください。

`-t worker`
 	
または
 	
`--tier worker`
	ワーカー環境を作成します。ウェブサーバー環境を作成する場合、このオプションを省略します。

`--timeout` *#*	コマンドがタイムアウトするまでの時間 (分) を設定します。
`--version` *version_label*	ローカルのプロジェクトディレクトリにあるアプリケーションのソースコードではなく、環境にデプロイするアプリケーションバージョンを指定します。 型: 文字列 有効な値: 既存のアプリケーションバージョンラベル
`--vpc`	環境に VPC を設定します。このオプションを指定すると、環境を起動する前に、必要なすべての設定を入力するよう求めるプロンプトが EB CLI によって表示されます。
`--vpc.dbsubnets` *subnet1,subnet2*	VPC 内のデータベースインスタンスのサブネットを指定します。`--vpc.id` が指定されている場合は必須です。
`--vpc.ec2subnets` *subnet1,subnet2*	VPC 内の Amazon EC2 インスタンスのサブネットを指定します。`--vpc.id` が指定されている場合は必須です。

名前	説明
`--vpc.elbpublic`	VPC のパブリックサブネットで Elastic Load Balancing ロードバランサーを起動します。 このオプションは `--tier worker` または `--single` オプションを使用して指定できません。
`--vpc.elbsubnets` *`subnet1,subnet2`*	VPC 内の Elastic Load Balancing ロードバランサーのサブネットを指定します。 このオプションは `--tier worker` または `--single` オプションを使用して指定できません。
`--vpc.id` *`ID`*	指定した VPC で環境を起動します。
`--vpc.publicip`	VPC 内のパブリックサブネットで Amazon EC2 インスタンスを起動します。 このオプションは `--tier worker` オプションを使用して指定できません。
`--vpc.securitygroups` *`securitygroup1,securitygroup2`*	セキュリティグループ ID を指定します。`--vpc.id` が指定されている場合は必須です。
一般的なオプション	

Output

成功すると、コマンドによって質問がいくつか表示され、作成オペレーションのステータスが返されます。起動時に問題があった場合は、eb events オペレーションを使用して詳細を取得できます。

アプリケーションで CodeBuild のサポートを有効にすると、eb create でコードの構築時に CodeBuild の情報が表示されます。Elastic Beanstalk での CodeBuild サポートの詳細については、「での EB CLI の使用 AWS CodeBuild」を参照してください。

例

以下の例では、インタラクティブモードで環境を作成しています。

```
$ eb create
Enter Environment Name
(default is tmp-dev): ENTER
Enter DNS CNAME prefix
(default is tmp-dev): ENTER
Select a load balancer type
1) classic
2) application
3) network
(default is 2): ENTER
Environment details for: tmp-dev
  Application name: tmp
  Region: us-east-2
  Deployed Version: app-141029_145448
  Environment ID: e-um3yfrzq22
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
  Updated: 2014-10-29 21:54:51.063000+00:00
Printing Status:
...
```

また、以下の例でも、インタラクティブモードで環境を作成しています。この例では、プロジェクトディレクトリにアプリケーションコードはありません。このコマンドはサンプルアプリケーションをデプロイし、ローカルプロジェクトディレクトリにダウンロードします。

```
$ eb create
Enter Environment Name
(default is tmp-dev): ENTER
Enter DNS CNAME prefix
(default is tmp-dev): ENTER
Select a load balancer type
1) classic
2) application
3) network
(default is 2): ENTER
NOTE: The current directory does not contain any source code. Elastic Beanstalk is
 launching the sample application instead.
```

```
Do you want to download the sample application into the current directory?
(Y/n): ENTER
INFO: Downloading sample application to the current directory.
INFO: Download complete.
Environment details for: tmp-dev
  Application name: tmp
  Region: us-east-2
  Deployed Version: Sample Application
  Environment ID: e-um3yfrzq22
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
  Updated: 2017-11-08 21:54:51.063000+00:00
Printing Status:
...
```

以下のコマンドは、いずれのプロンプトも表示せずに環境を作成します。

```
$ eb create dev-env
Creating application version archive "app-160312_014028".
Uploading test/app-160312_014028.zip to S3. This may take a while.
Upload Complete.
Application test has been created.
Environment details for: dev-env
  Application name: test
  Region: us-east-2
  Deployed Version: app-160312_014028
  Environment ID: e-6fgpkjxyyi
  Platform: 64bit Amazon Linux 2015.09 v2.0.8 running PHP 5.6
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-03-12 01:40:33.614000+00:00
Printing Status:
...
```

以下のコマンドは、カスタム VPC に環境を作成します。

```
$ eb create dev-vpc --vpc.id vpc-0ce8dd99 --vpc.elbsubnets subnet-
b356d7c6,subnet-02f74b0c --vpc.ec2subnets subnet-0bb7f0cd,subnet-3b6697c1 --
vpc.securitygroup sg-70cff265
Creating application version archive "app-160312_014309".
Uploading test/app-160312_014309.zip to S3. This may take a while.
Upload Complete.
```

```
Environment details for: dev-vpc
  Application name: test
  Region: us-east-2
  Deployed Version: app-160312_014309
  Environment ID: e-pqkcip3mns
  Platform: 64bit Amazon Linux 2015.09 v2.0.8 running Java 8
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-03-12 01:43:14.057000+00:00
Printing Status:
...
```

eb deploy

説明

初期化されたプロジェクトディレクトリから実行アプリケーションにアプリケーションソースバンドルをデプロイします。

git がインストールされている場合、EB CLI は `git archive` コマンドを使用して最新の `.zip` コマンドのコンテンツから `git commit` ファイルを作成します。

ただし、プロジェクトディレクトリに `.ebignore` が存在する場合、EB CLI は git コマンドおよびセマンティクスを使用せずにソースバンドルを作成します。つまり、EB CLI は `.ebignore` で指定されたファイルを無視し、他のすべてのファイルを含めます。特に、コミットされていないソースファイルが含まれます。

> **Note**
>
> プロジェクトフォルダの ZIP ファイルを作成する代わりにビルドプロセスからの中間生成物をデプロイするように EB CLI を設定できます。詳細については、「プロジェクトフォルダの代わりにアーティファクトをデプロイする」を参照してください。

構文

eb deploy

eb deploy ***environment-name***

オプション

名前	説明
`-l` *`version_label`* または `--label` *`version_label`*	EB CLI で作成するバージョンに使用するラベルを指定します。ラベルが既に使用されている場合、EB CLI はそのラベルで以前のバージョンを再デプロイします。 型: 文字列
`--env-group-suffix` *`groupname`*	環境の名前に連結するグループ名。環境を構成するのみで使用します。

`-m "`*`version_description`*`"`
 	
または
 	
`--message "`*`version_description`*`"`
	アプリケーションバージョンの説明。二重引用符で囲みます。
	
	型: 文字列

`--modules` *`component-a component-b`*	更新するコンポーネントのリスト。環境を構成するのみで使用します。

`-p`
 	
または
 	
`--process`
	ソースバンドル内の環境マニフェストおよび設定ファイルを事前処理し、検証します。設定ファイルを検証することで、アプリケーションバージョンを環境にデプロイする前に問題を特定できます。

`--source codecommit/` *`repository-name/branch-name`*	CodeCommit リポジトリとブランチ。
`--staged`	HEAD コミットではなく Git インデックスでステージングされたデプロイファイル。
`--timeout` *`#`*	コマンドがタイムアウトするまでの時間 (分) 。

名前	説明
`--version` *`version_label`*	デプロイする既存のアプリケーションバージョン。 型: 文字列
一般的なオプション	

Output

成功すると、コマンドは `deploy` オペレーションのステータスを返します。

アプリケーションで CodeBuild のサポートを有効にすると、eb deploy でコードの構築時に CodeBuild の情報が表示されます。Elastic Beanstalk での CodeBuild サポートの詳細については、「での EB CLI の使用 AWS CodeBuild」を参照してください。

例

以下の例は、現在のアプリケーションをデプロイします。

```
$ eb deploy
2018-07-11 21:05:22    INFO: Environment update is starting.
2018-07-11 21:05:27    INFO: Deploying new version to instance(s).
2018-07-11 21:05:53    INFO: New application version was deployed to running EC2
 instances.
2018-07-11 21:05:53    INFO: Environment update completed successfully.
```

eb events

説明

環境の最新のイベントを返します。

root ディレクトリにカスタムプラットフォームを指定する `platform.yaml` ファイルが含まれている場合、このコマンドでもビルダー環境の最新のイベントを返します。

構文

eb events

eb events ***environment-name***

オプション

名前	説明

`-f`
 	
または
 	
`--follow`
	イベントをストリーミングします。キャンセルするには、Ctrl +C を押します。

Output

成功すると、コマンドは最新のイベントを返します。

例

以下の例は、最新の 個のイベントを返します。

```
$ eb events
2014-10-29 21:55:39     INFO    createEnvironment is starting.
2014-10-29 21:55:40     INFO    Using elasticbeanstalk-us-west-2-111122223333 as Amazon
 S3 storage bucket for environment data.
2014-10-29 21:55:57     INFO    Created load balancer named: awseb-e-r-AWSEBLoa-
NSKUOK5X6Z9J
2014-10-29 21:56:16     INFO    Created security group named: awseb-e-rxgrhjr9bx-stack-
AWSEBSecurityGroup-1UUHU5LZ20ZY7
2014-10-29 21:57:18     INFO    Waiting for EC2 instances to launch. This may take a
 few minutes.
2014-10-29 21:57:18     INFO    Created Auto Scaling group named: awseb-e-rxgrhjr9bx-
stack-AWSEBAutoScalingGroup-1TE320ZCJ9RPD
2014-10-29 21:57:22     INFO    Created Auto Scaling group policy named:
 arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:2cced9e6-859b-421a-
be63-8ab34771155a:autoScalingGroupName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingGroup-1TE320ZCJ9RPD:policyName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingScaleUpPolicy-1I2ZSNVU4APRY
2014-10-29 21:57:22     INFO    Created Auto Scaling group policy named:
 arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:1f08b863-
bf65-415a-b584-b7fa3a69a0d5:autoScalingGroupName/awseb-e-rxgrhjr9bx-stack-
```

```
AWSEBAutoScalingGroup-1TE320ZCJ9RPD:policyName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingScaleDownPolicy-1E3G7PZKZPSOG
2014-10-29 21:57:25     INFO    Created CloudWatch alarm named: awseb-e-rxgrhjr9bx-
stack-AWSEBCloudwatchAlarmLow-VF5EJ549FZBL
2014-10-29 21:57:25     INFO    Created CloudWatch alarm named: awseb-e-rxgrhjr9bx-
stack-AWSEBCloudwatchAlarmHigh-LA9YEW3O6WJO
2014-10-29 21:58:50     INFO    Added EC2 instance 'i-c7ee492d' to Auto ScalingGroup
 'awseb-e-rxgrhjr9bx-stack-AWSEBAutoScalingGroup-1TE320ZCJ9RPD'.
2014-10-29 21:58:53     INFO    Successfully launched environment: tmp-dev
2014-10-29 21:59:14     INFO    Environment health has been set to GREEN
2014-10-29 21:59:43     INFO    Adding instance 'i-c7ee492d' to your environment.
```

eb health

説明

環境の最新のヘルスステータスを返します。

root ディレクトリにカスタムプラットフォームを指定する `platform.yaml` ファイルが含まれている場合、このコマンドはビルダー環境の最新のヘルスを返します。

構文

eb health

eb health ***environment-name***

オプション

名前	説明

`-r`
または
`--refresh`
	ヘルス情報を対話モードで表示します。レポートされる情報は 10 秒ごとに更新されます。

`--mono`	出力に色を表示しません。

Output

成功した場合、コマンドは最新のヘルスステータスを返します。

例

次の例では、Linux 環境の最新のヘルス情報を返します。

```
~/project $ eb health
 elasticBeanstalkExa-env                                  Ok
 2015-07-08 23:13:20
WebServer
 Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0

  instance-id   status     cause
                                        health
    Overall     Ok
  i-d581497d    Ok
  i-d481497c    Ok
  i-136e00c0    Ok
  i-126e00c1    Ok
  i-8b2cf575    Ok

  instance-id   r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75     p50
   p10                                  requests
    Overall     671.8   100.0    0.0    0.0    0.0    0.003    0.002    0.001   0.001
 0.000
  i-d581497d    143.0    1430      0      0      0    0.003    0.002    0.001   0.001
 0.000
  i-d481497c    128.8    1288      0      0      0    0.003    0.002    0.001   0.001
 0.000
  i-136e00c0    125.4    1254      0      0      0    0.004    0.002    0.001   0.001
 0.000
  i-126e00c1    133.4    1334      0      0      0    0.003    0.002    0.001   0.001
 0.000
  i-8b2cf575    141.2    1412      0      0      0    0.003    0.002    0.001   0.001
 0.000

  instance-id   type       az   running     load 1  load 5      user%  nice%  system%
 idle%   iowait%                              cpu
```

```
 i-d581497d    t2.micro   1a   12 mins        0.0    0.04        6.2    0.0      1.0
92.5       0.1
 i-d481497c    t2.micro   1a   12 mins       0.01    0.09        5.9    0.0      1.6
92.4       0.1
 i-136e00c0    t2.micro   1b   12 mins       0.15    0.07        5.5    0.0      0.9
93.2       0.0
 i-126e00c1    t2.micro   1b   12 mins       0.17    0.14        5.7    0.0      1.4
92.7       0.1
 i-8b2cf575    t2.micro   1c   1 hour        0.19    0.08        6.5    0.0      1.2
92.1       0.1

 instance-id   status     id   version              ago
                                   deployments
 i-d581497d    Deployed   1    Sample Application   12 mins
 i-d481497c    Deployed   1    Sample Application   12 mins
 i-136e00c0    Deployed   1    Sample Application   12 mins
 i-126e00c1    Deployed   1    Sample Application   12 mins
 i-8b2cf575    Deployed   1    Sample Application   1 hour
```

eb init

説明

一連の質問を表示し、それに応答することによって、EB CLI で作成された Elastic Beanstalk アプリケーションのデフォルト値を設定します。

> **Note**
>
> で設定した値は、現在のコンピュータの現在のディレクトリとリポジトリeb initに適用されます。
> コマンドは、アカウントに Elastic Beanstalk アプリケーションを作成します。Elastic Beanstalk 環境を作成するには、eb init を実行した後に eb create を実行します。

構文

eb init

eb init *application-name*

オプション

`--platform` オプションを指定せずに eb init を実行すると、EB CLI は各設定の値の入力を求めるプロンプトを表示します。

> **Note**
>
> eb init を使用して新しいキーペアを作成するには、`ssh-keygen` がローカルマシンにインストールされており、コマンドラインから呼び出せる必要があります。

名前	説明	

`-i`
 	
`--interactive`
	すべての eb init コマンドオプションの値を指定するように求める指示が、EB CLI で必ず表示されます。
	
	Note
	`init` コマンドを使用すると、（デフォルト）値が設定されていない eb init コマンドオプションに値を指定するように求められます。ディレクトリで eb init コマンドを初めて実行したとき、EB CLI では、コマンドオプションに値を指定するように求められない場合があります。そのような場合は、すでに設定されている内容を変更するために、`--interactive` オプションを使用します。	

`-k` *`keyname`* `--keyname` *`keyname`*	Elastic Beanstalk アプリケーションを実行する Amazon EC2 インスタンスに安全にログインするために、Secure Shell (SSH) クライアントで使用する Amazon EC2 キーペアの名前。	
`--modules` *`folder-1 folder-2`*	初期化する子ディレクトリのリスト。環境を構成するのみで使用します。	

名前	説明	
-p *platform-version* --platform *platform-version*	使用するプラットフォームバージョン。プラットフォーム、プラットフォームとバージョン、プラットフォームブランチ、ソリューションスタック名、またはソリューションスタック ARN を指定できます。次に例を示します。 • php、PHP、node.js – 指定されたプラットフォームの最新のプラットフォームバージョン • php-7.2、"PHP 7.2" – 推奨される (通常は最新の) PHP 7.2 プラットフォームバージョン • "PHP 7.2 running on 64bit Amazon Linux" – このプラットフォームブランチに推奨される (通常は最新の) PHP プラットフォームバージョン • "64bit Amazon Linux 2017.09 v2.6.3 running PHP 7.1" – ソリューションスタック名で指定された PHP プラットフォームバージョン • "arn:aws:elasticbeanstalk:us-east-2::platform/PHP 7.1 running on 64bit Amazon Linux/2.6.3" – ソリューションスタック ARN で指定された PHP プラットフォームバージョン 利用可能な設定のリストを取得するには、eb platform list を使用します。 --platform オプションを指定して、インタラクティブ設定をスキップします。 **Note** このオプションを使用すると、EB CLI は、他のすべてのオプションについて値の入力を要求しません。ただし、各オプションのデフォルト値の使用が前提となります。デフォルト	

名前	説明	
	値を使用しない場合は、そのオプションの値を指定してください。	
`--source codecommit/ repository-name/branch-name`	CodeCommit リポジトリとブランチ。	
`--tags key1=value1`	アプリケーションにタグを付けます。タグは、`key=value` ペアのカンマ区切りリストとして指定されます。 詳細については、「アプリケーションのタグ付け」を参照してください。	
一般的なオプション		

CodeBuild サポート

buildspec.yml ファイルを含むフォルダで eb init を実行すると、Elastic Beanstalk はファイルを解析して、Elastic Beanstalk に固有のオプションを持つ eb_codebuild_settings エントリを特定します。Elastic Beanstalk での CodeBuild サポートの詳細については、「での EB CLI の使用 AWS CodeBuild」を参照してください。

Output

成功すると、コマンドは一連のプロンプトで新しい Elastic Beanstalk アプリケーションをセットアップする手順を示します。

例

次のリクエストの例は、EB CLI を初期化し、アプリケーションに関する情報を入力するように求めます。*########*テキストを独自の値に置き換えます。

```
$ eb init -i
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
```

```
3) us-west-2 : US West (Oregon)
4) eu-west-1 : Europe (Ireland)
5) eu-central-1 : Europe (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
...
(default is 3): 3

Select an application to use
1) HelloWorldApp
2) NewApp
3) [ Create new Application ]
(default is 3): 3

Enter Application Name
(default is "tmp"):
Application tmp has been created.

It appears you are using PHP. Is this correct?
(y/n): y

Select a platform branch.
1) PHP 7.2 running on 64bit Amazon Linux
2) PHP 7.1 running on 64bit Amazon Linux (Deprecated)
3) PHP 7.0 running on 64bit Amazon Linux (Deprecated)
4) PHP 5.6 running on 64bit Amazon Linux (Deprecated)
5) PHP 5.5 running on 64bit Amazon Linux (Deprecated)
6) PHP 5.4 running on 64bit Amazon Linux (Deprecated)
(default is 1): 1
Do you want to set up SSH for your instances?
(y/n): y

Select a keypair.
1) aws-eb
2) [ Create new KeyPair ]
(default is 2): 1
```

eb labs

説明

eb labs サポート作業の進行状況または実験機能のサブコマンド。これらのコマンドは、EB CLI の今後のバージョンで削除されるか改変される場合があり、将来の互換性は保証されません。

使用できるサブコマンドのリストと説明については、eb labs --help を実行してください。

eb list

説明

現在のアプリケーションのすべての環境を一覧表示します。`--all` オプションを指定した場合は、すべてのアプリケーションのすべての環境を一覧表示します。

root ディレクトリにカスタムプラットフォームを指定する `platform.yaml` ファイルが含まれている場合、このコマンドはビルダー環境に関する情報も一覧表示します。

構文

eb list

オプション

名前	説明

`-a`
または
`--all`
	すべてのアプリケーションのすべての環境を一覧表示します。

`-v`
または
`--verbose`
	インスタンスを含む、すべての環境に関するより詳細な情報を表示します。

名前	説明
一般的なオプション	

Output

成功すると、コマンドは環境名のリストを返します。現在の環境にはアスタリスク (*) が付いています。

例 1

次の例では、環境名を一覧表示しており、tmp-dev がデフォルトの環境であることを示しています。

```
$ eb list
* tmp-dev
```

例 2

次の例では、詳細情報を付加して環境を一覧表示しています。

```
$ eb list --verbose
Region: us-west-2
Application: tmp
    Environments: 1
        * tmp-dev : ['i-c7ee492d']
```

eb local

説明

eb local run を使用すると、アプリケーションのコンテナが Docker 内のローカルで実行されます。eb local status を使用すると、アプリケーションのコンテナのステータスを確認できます。eb local open を使用すると、アプリケーションがウェブブラウザで開きます。eb local logs を使用すると、アプリケーションのログの場所を取得できます。

eb local setenv および eb local printenv を使用すると、eb local run を使用してローカルで実行した Docker コンテナに渡される環境変数を設定、表示できます。

すべての eb local コマンドは、eb init を使用して EB CLI リポジトリとして初期化された Docker アプリケーションのプロジェクトディレクトリで実行する必要があります。

> **Note**
>
> Linux または macOS を実行しているローカルコンピュータで eb local を使用します。このコマンドでは Windows はサポートされません。
> このコマンドを macOS で使用する前に、Mac 用の Docker をインストールすると共に、boot2docker がインストールされていない (または実行パスに含まれていない) ことを確認してください。eb local コマンドは boot2docker (ある場合) を使用しようとしますが、macOS では正常に機能しません。

構文

eb local run

eb local status

eb local open

eb local logs

eb local setenv

eb local printenv

オプション

eb local run

名前	説明
`--envvars` *`key1=value1,key2=value2`*	EB CLI がローカルの Docker コンテナに渡す環境変数を設定します。複数コンテナ環境では、すべての変数がすべてのコンテナに渡されます。
`--port` *`hostport`*	ホストのポートをコンテナの公開ポートにマッピングします。このオプションを指定しない場合、EB CLI は、ホストとコンテナの両方で同じポートを使用します。

名前	説明
	このオプションは、Docker プラットフォームアプリケーションでのみ機能します。Multicontainer Docker プラットフォームには適用されません。
一般的なオプション	

eb local status

eb local open

eb local logs

eb local setenv

eb local printenv

名前	説明
一般的なオプション	

Output

eb local run

Docker からのステータスメッセージ。アプリケーションが実行されている間は、アクティブなままです。アプリケーションを停止するには、Ctrl+C を押します。

eb local status

アプリケーションで使用される各コンテナのステータス (実行中かどうか) 。

eb local open

アプリケーションをウェブブラウザで開き、終了します。

eb local logs

eb local run を使用してローカルで実行されたアプリケーションによって、プロジェクトディレクトリに生成されるログの場所。

eb local setenv

なし

eb local printenv

eb local setenv を使用して設定された環境変数の名前と値。

例

eb local run

```
~/project$ eb local run
Creating elasticbeanstalk_phpapp_1...
Creating elasticbeanstalk_nginxproxy_1...
Attaching to elasticbeanstalk_phpapp_1, elasticbeanstalk_nginxproxy_1
phpapp_1     | [23-Apr-2015 23:24:25] NOTICE: fpm is running, pid 1
phpapp_1     | [23-Apr-2015 23:24:25] NOTICE: ready to handle connections
```

eb local status

ローカルのコンテナのステータスを表示します。

```
~/project$ eb local status
Platform: 64bit Amazon Linux 2014.09 v1.2.1 running Multi-container Docker 1.3.3
 (Generic)
Container name: elasticbeanstalk_nginxproxy_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): 80
Full local URL(s): 127.0.0.1:80

Container name: elasticbeanstalk_phpapp_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): None
Full local URL(s): None
```

eb local logs

現在のプロジェクトのログのパスを表示します。

```
~/project$ eb local logs
```

```
Elastic Beanstalk will write logs locally to /home/user/project/.elasticbeanstalk/logs/
local.
Logs were most recently created 3 minutes ago and written to /home/user/
project/.elasticbeanstalk/logs/local/150420_234011665784.
```

eb local setenv

eb local run とともに使用する環境変数を設定します。

```
~/project$ eb local setenv PARAM1=value
```

eb local setenv を使用して設定する環境変数を出力します。

```
~/project$ eb local printenv
Environment Variables:
PARAM1=value
```

eb logs

説明

eb logs コマンドには、CloudWatch Logs へのログストリーミングの有効化または無効化と、インスタンスログまたは CloudWatch Logs ログの取得という 2 つの明確な目的があります。`--cloudwatch-logs` (`-cw`) オプションを使用すると、コマンドはログストリーミングを有効または無効にします。このオプションを使用しない場合は、ログを取得します。

ログを取得するときは、完全なログを取得する `--all`、`--zip`、または `--stream` オプションを指定します。これらのオプションを指定しない場合は、Elastic Beanstalk は末尾のログを取得します。

コマンドは、指定された環境またはデフォルト環境のログを処理します。関連するログは、コンテナタイプに応じて異なります。root ディレクトリにカスタムプラットフォームを指定する `platform.yaml` ファイルが含まれている場合、このコマンドはビルダー環境のログも処理します。

詳細については、「the section called “CloudWatch Logs”」を参照してください。

構文

CloudWatch Logs へのログストリーミングを有効または無効にするには:

```
eb logs --cloudwatch-logs [enable | disable] [--cloudwatch-log-source instance |
 environment-health | all] [environment-name]
```

インスタンスログを取得するには:

```
eb logs [-all | --zip | --stream] [--cloudwatch-log-source instance] [--
instance instance-id] [--log-group log-group] [environment-name]
```

環境のヘルスログを取得するには:

```
eb logs [-all | --zip | --stream] --cloudwatch-log-source environment-health
 [environment-name]
```

オプション

名前	説明

`-cw [enable | disable]`
 	
または
 	
`--cloudwatch-logs [enable | disable]`
	CloudWatch Logs へのログストリーミングを有効または無効にします。引数を指定しない場合、ログストリーミングが有効になります。さらに `--cloudwatch-log-source` (`-cls`) オプションが指定されていない場合は、インスタンスのログストリーミングが有効または無効になります。

`-cls instance | environment-health | all`
 	
または
 	
`--cloudwatch-log-source instance | environment-health | all`
	CloudWatch Logs を使用しているときにログのソースを指定します。コマンドの有効または無効の形式では、CloudWatch Logs ストリーミングを有効または無効にするログになります。コマンドの検索形式では、これらは CloudWatch Logs から検索するログです。
	
	有効な値:
	
	• `--cloudwatch-logs` (有効または無効にする) - `instance | environment-health | all`
	• `--cloudwatch-logs` なし (取得) - `instance | environment-health`
	
	値の意味は次のとおりです。

名前	説明
	• `instance` (デフォルト) - インスタンスログ • `environment-health` - 環境ヘルスログ (環境で拡張ヘルスが有効になっている場合のみサポートされます) • `all` - 両方のログソース

`-a`
または
`--all`
	すべてのログを取得し、それらのログを `.elasticbeanstalk/logs` ディレクトリに保存します。

`-z`
または
`--zip`
	すべてのログを取得し、`.zip` ファイルに圧縮して、そのファイルを `.elasticbeanstalk/logs` ディレクトリに保存します。

`--stream`	完全なログをストリーミングします (継続的な出力)。このオプションを使用すると、コマンドは中断されるまで (**Ctrl+C** を押す) 実行を続けます。
`-i` *`instance-id`* または `--instance` *`instance-id`*	指定したインスタンスのみのログを取得します。

名前	説明
`-g` *`log-group`* または `--log-group` *`log-group`*	ログの取得元となる CloudWatch Logs ロググループを指定します。このオプションは、CloudWatch Logs へのインスタンスログストリーミングが使用可能になっている場合にのみ有効です。 インスタンスログストリーミングが有効になっていて `--log-group` オプションを指定しない場合、デフォルトのロググループは次のいずれかです。 • Amazon Linux 2 – `/aws/elasticbeanstalk/` *`environment-name`* `/var/log/eb-engine.log` • Windows プラットフォーム - `/aws/elasticbeanstalk/` *`environment-name`* `/EBDeploy-Log` • Amazon Linux AMI (AL1) – `/aws/elasticbeanstalk/` *`environment-name`* `/var/log/eb-activity.log` **Note** 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。 各ログファイルに対応するロググループの詳細については、「Elastic Beanstalk が CloudWatch Logs を設定する方法」を参照してください。
一般的なオプション	

Output

デフォルトでは、ログはターミナルに直接表示されます。ページングプログラムを使用して出力を表示します。**Q** または **q** を押して終了します。

`--stream` では、ターミナルの既存のログを表示し、実行を続けます。**Ctrl+C** を押して終了します。

`--all` と `--zip` を使用して、ログをローカルファイルに保存し、ファイルの場所を表示します。

例

次の例では、CloudWatch Logs へのインスタンスログストリーミングを有効にします。

```
$ eb logs -cw enable
Enabling instance log streaming to CloudWatch for your environment
After the environment is updated you can view your logs by following the link:
https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#logs:prefix=/aws/
elasticbeanstalk/environment-name/
Printing Status:
2018-07-11 21:05:20    INFO: Environment update is starting.
2018-07-11 21:05:27    INFO: Updating environment environment-name's configuration
 settings.
2018-07-11 21:06:45    INFO: Successfully deployed new configuration to environment.
```

次の例では、インスタンスログを `.zip` ファイルに取得します。

```
$ eb logs --zip
Retrieving logs...
Logs were saved to /home/workspace/environment/.elasticbeanstalk/logs/150622_173444.zip
```

eb migrate

説明

インターネットインフォメーションサービス (IIS) サイトとアプリケーションを Windows サーバーから Elastic Beanstalk に移行します。コマンドはアプリケーションをパッケージ化し、設定を保持して、新しい Elastic Beanstalk 環境にデプロイします。

IIS サイトとアプリケーションの移行の詳細については、「」を参照してください[IIS アプリケーションの移行](#)。

> **Note**
>
> このコマンドを使用する前に、システムが次の要件を満たしていることを確認してください。
>
> - Internet Information Services (IIS) バージョン 7.0 以降
> - Web Deploy 3.6 以降がインストールされている
> - Windows サーバーの管理者権限
> - AWS 適切なアクセス許可で設定された 認証情報
> - ソースサーバーには、 AWS サービスへのアウトバウンドインターネットアクセスがあります。

次の手順は、移行プロセスをまとめたものです。

1. IIS サイトとその設定を検出します。
2. アプリケーションのコンテンツと設定をパッケージ化します。
3. Elastic Beanstalk 環境とアプリケーションを作成します。
4. 保存された設定でアプリケーションをデプロイします。

コマンドは、次のリストに示すように、構造化ディレクトリに移行アーティファクトを作成します。

```
C:\migration_workspace\
### .\migrations\latest\
    ### upload_target.zip
    ### upload_target\
        ### [SiteName].zip                      # One ZIP per default application of
 IIS site
        ### [SiteName-ApplicationName].zip      # One ZIP per additional application
        ### aws-windows-deployment-manifest.json
        ### ebmigrateScripts\
            ### site_installer.ps1              # Site installation scripts
            ### permission_handler.ps1          # Permission management
            ### other helper scripts
```

eb migrate cleanup を使用して、これらのアーティファクトを管理します。

構文

eb migrate ***[options]***

eb migrate explore ***[options]***

eb migrate cleanup ***[options]***

引数なしで実行すると、 は非インタラクティブモードでeb migrate動作します。インタラクティブモードで実行するには、 を実行しますeb migrate --interactive。

インタラクティブモードコマンドは、次の情報を求めます。

- 移行する IIS サイトの選択
- 環境名とアプリケーション名
- プラットフォームバージョンの選択
- インスタンスタイプおよびその他の設定オプション

サブコマンド

探索

eb migrate explore サブコマンドは IIS サーバーを調べ、利用可能なサイトを一覧表示します。

このコマンドを使用して、次の情報を表示します。

- サーバー上のすべての IIS サイトを表示する
- を使用して`--verbose`、以下を含む詳細な設定を検査します。
  - サイトバインディングとポート
  - アプリケーションプール
  - 仮想ディレクトリとその物理パス
  - [認証] 設定

```
PS C:\migrations_workspace >  eb migrate explore
Default Web Site
```

```
Site2
site3
router
```

```
PS C:\migrations_workspace >  eb migrate explore --verbose
1: Default Web Site:
  - Bindings:
    - *:8083:
  - Application '/':
    - Application Pool: DefaultAppPool
    - Enabled Protocols: http
    - Virtual Directories:
      - /:
        - Physical Path: C:\inetpub\wwwroot
        - Logon Method: ClearText
  - Application '/dotnet-6-0':
    - Application Pool: DefaultAppPool
    - Enabled Protocols: http
    - Virtual Directories:
      - /:
        - Physical Path: C:\inetpub\AspNetCoreWebApps\CoreWebApp-6-0
        - Logon Method: ClearText
  - Application '/dotnet-8-0':
    - Application Pool: DefaultAppPool
    - Enabled Protocols: http
    - Virtual Directories:
      - /:
        - Physical Path: C:\inetpub\AspNetCoreWebApps\CoreWebApp-8-0
        - Logon Method: ClearText
2: Site2:
  - Bindings:
    - *:8081:
...
```

クリーンアップ

eb migrate cleanup サブコマンドは、次のアクションを使用して移行アーティファクトを管理します。

- で最後に成功した移行の保存 `./migrations/latest`
- 古い移行ディレクトリの削除
- 重要な設定ファイルの保守

```
PS C:\migrations_workspace >   eb migrate cleanup
Are you sure you would like to cleanup older artifacts within `./migrations/`? (y/N):
```

クリーンアップ中に確認プロンプトをスキップ`--force`するには、 を使用します。

```
PS C:\migrations_workspace >   eb migrate cleanup --force
```

オプション

これらのオプションはいずれも必須ではありません。オプションeb migrateなしで を実行すると、EB CLI は非インタラクティブモードで実行されます。ではeb migrate --interactive、EB CLI から必要な設定の値を入力または選択するように求められます。

名前	説明
`-a` *`application-name`* or `--application-name` *`application-name`*	新しい Elastic Beanstalk アプリケーションの名前。 タイプ: 文字列 デフォルト: EBMigratedApp
`--archive` *`directory-or-zip`*	によって以前に生成されたソースコードを含むディレクトリまたは ZIP ファイルeb migrate --archive-only。 このオプションを使用して、以前に作成した移行パッケージをデプロイします。 例: `--archive .\migrations\latest\upload_target` または `--archive .\migrations\latest\upload_target.zip`

`-ao`
 	
or
 	
`--archive-only`
	デプロイなしで送信先アーカイブディレクトリのみを作成します。
	
	結果のディレクトリは、 `archive`オプションまたは eb migrateを使用して手動でデプロイできますeb deploy。

名前	説明
`-c` *`subdomain-name`* または `--cname` *`subdomain-name`*	移行されたアプリケーションの CNAME DNS エントリのプレフィックスとなるサブドメイン名。 タイプ: 文字列 デフォルト: 環境の名前

`-cf`
 	
or
 	
`--copy-firewall-config`
	アクティブなバインディングを持つすべての HTTP ポートの送信先にソースサーバーのファイアウォール設定をコピーします。
	
	対応するセキュリティグループルールを作成します AWS。

`-es` *`snapshot-id`* [*`snapshot-id`* ...] or `--ebs-snapshots` *`snapshot-id`* [*`snapshot-id`* ...]	環境に関連付ける Amazon EBS スナップショット IDs のカンマ区切りリスト。 例: `--ebs-snapshots snap-1234567890abcdef0, snap-0987654321fedcba1`
`--encrypt-ebs-volumes`	すべての新しい Amazon EBS ボリュームに暗号化を適用します。 **Important** これは、今後のすべての Amazon EBS ボリュームの作成に影響するアカウント全体の設定です。

名前	説明
`-e` *`environment-name`* or `--environment-name` *`environment-name`*	新しい Elastic Beanstalk 環境の名前。 タイプ: 文字列 デフォルト: EBMigratedEnv 制約: 4 ~ 40 文字の長さにする必要があります。文字、数字、ハイフンのみを含めることができます。ハイフンでスタートまたは終了することはできません。
`--force`	オペレーション中に確認プロンプトをスキップします。 cleanup サブコマンドで使用すると、 は確認なしで移行アーティファクトを削除します。
`-ip` *`profile-name`* or `--instance-profile` *`profile-name`*	環境の Amazon EC2 インスタンスに関連付けるインスタンスプロファイル。 指定しない場合、 は Elastic Beanstalk リソースにアクセスするアクセス許可を持つデフォルトのインスタンスプロファイルを作成します。詳細については、「the section called “インスタンスプロファイル”」を参照してください。
`-i` *`########`* or `--instance-type` *`########`*	Elastic Beanstalk 環境の Amazon EC2 インスタンスタイプ。 タイプ: 文字列 デフォルト: c5.2xlarge 使用可能なインスタンスタイプについては、Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンスタイプ」を参照してください。 Amazon EC2

名前	説明

`-in`
 	
or
 	
`--interactive`
	移行プロセスの強制インタラクティブモード。
	
	デフォルトが使用可能な場合でも、設定値の入力を求めます。

`-k` *`###`* or `--keyname` *`###`*	環境インスタンスへの RDP アクセスを有効にする Amazon EC2 キーペア。 ログに表示されないインスタンスレベルの問題の調査に役立ちます。 有効な値: Amazon EC2 に登録されている既存のキーペア名
`-p` *`platform-version`* または `--platform` *`platform-version`*	環境の Elastic Beanstalk プラットフォームランタイム。指定しない場合、ホスト Windows Server バージョンから自動的に検出されます。 例: `"64bit Windows Server 2016 v2.16.2 running IIS 10.0"` 利用可能なプラットフォームバージョンのリストについては、eb プラットフォームリストを使用します。

名前	説明
`--remote`	リモートモードで移行を実行することを示します。このオプションを使用すると、踏み台ホストからの実行が可能になります。踏み台ホストは、Elastic Beanstalk に移行するアプリケーションと設定を含むターゲットサーバーに接続します。踏み台サーバーから実行すると、 は設定`eb migrate`を検出し、踏み台ホストで移行ロジックをステージングしてから、アプリケーションを新しい Elastic Beanstalk 環境にデプロイします。 このオプションを使用すると、移行する必要がある Windows サーバーに EB CLI と Python をインストールする必要がなくなります。代わりに踏み台ホストに Python と EB CLI をインストールします。ここでは、eb migrate コマンドを `--remote`オプションで実行します。`--target-ip` オプションを使用して、移行する IIS 設定でホストを指定します。 `--target-ip` 、、`--username` および で使用する必要があります`--password` 。
`--target-ip` *`ip-address`*	移行する IIS サーバーを含むリモート Windows マシンのパブリック IP アドレス。 を使用する場合に必要です`--remote`。を使用する場合にのみ指定できます`--remote`。
`--username` *`username`*	移行する IIS サーバーを含むリモート Windows マシンにアクセスするためのユーザープロファイルのユーザー名。 を使用する場合に必要です`--remote`。を使用する場合にのみ指定できます`--remote`。

名前	説明
`--password` *`password`*	移行する IIS サーバーを含むリモート Windows マシンにアクセスするためのユーザープロファイルのパスワード。 を使用する場合に必要です`--remote`。を使用する場合にのみ指定できます`--remote`。
`-sr` *`role-name`* or `--service-role` *`role-name`*	Elastic Beanstalk が関連サービスを管理するための IAM AWS サービスロール。 指定しない場合、 は必要なアクセス許可を持つデフォルトのサービスロールを作成します。詳細については、「the section called “サービスロール”」を参照してください。 **Note** 完全な ARN ではなく、ロール名のみを指定します。Elastic Beanstalk は完全な ARN を自動的に作成します。
`-s` *`site-names`* or `--sites` *`site-names`*	移行する IIS サイトのカンマ区切りリスト。指定しない場合、 はサーバー上の利用可能なすべてのサイトを移行します。 例: `--sites "Default Web Site,Intranet,API"`
`--ssl-certificates` *`certificate-arn`* [,*`certificate-arn`* ...]	Application Load Balancer に関連付ける ACM SSL 証明書 ARNs のカンマ区切りリスト。 HTTPS バインディングを使用してサイトを移行する場合に必要です。 例: `--ssl-certificates arn:aws:acm:region:account:certificate/certificate-id`

名前	説明
-t *key1*=*value1*[,*key2*=*value2* .. or --tags *key1*=*value1*[,*key2*=*va*	環境内の新しいリソースにタグを付けるためのkey=value ペアのカンマ区切りリスト: 環境、Elastic Beanstalk アプリケーション、アプリケーションバージョン。 詳細については、「環境のタグ付け」を参照してください。
--verbose	移行プロセス中に詳細情報を表示します。 explore サブコマンドとともに使用すると、 は包括的なサイト設定の詳細を表示します。

名前	説明

`-vpc` *`config-file-or-string`*
 	
or
 	
`--vpc-config` *`config-file-or-string`*
	環境の VPC 設定。JSON ファイルパスまたは JSON 文字列として指定します。
	
	設定には以下を含める必要があります。
	
	`{`
	`  "id": "vpc-1234567890abcdef0",`
	`  "publicip": "true|false",`
	`  "elbscheme": "public|private",`
	`  "ec2subnets": ["subnet-a1b2c3d4",`
	`  "subnet-e5f6g7h8"],`
	`  "securitygroups": "sg-123456,sg-789012",`
	`  "elbsubnets": ["subnet-a1b2c3d4",`
	`  "subnet-e5f6g7h8"]`
	`}`
	
	• `id`: (必須) VPC 識別子
	• `publicip`: インスタンスにパブリック IPsを割り当てるかどうか
	• `elbscheme` ：ロードバランサースキーム (パブリックまたはプライベート)
	• `ec2subnets` ：EC2 インスタンスIDs のリスト
	• `securitygroups` ：カンマ区切りのセキュリティグループ IDs
	• `elbsubnets` ：ロードバランサーのサブネット IDsのリスト
	
	Important
	パラメータを指定すると、移行はソース環境からの既存の VPC 設定を無視します`--vpc-config` 。このパラメータを使用すると、移行では、渡す設定ファイルで指定された VPC 設定のみが使用されます。このパラメータを使用すると、ソースインスタンスの VPC 設定を検出する

名前	説明
	か、デフォルトの VPC を使用するというデフォルトの動作が上書きされます。
一般的なオプション	

Output

コマンドは、移行プロセス全体でステータスの更新を提供します。

1. VPC 設定検出 (EC2 インスタンスで実行されている場合)
2. 各サイトのソースバンドル生成の進行状況
3. 環境作成ステータス
4. デプロイの進捗状況

成功すると、以下を含む新しい環境の詳細が表示されます。

- 環境名と ID
- アプリケーション名
- リージョン
- プラットフォームバージョン
- 環境 CNAME

移行中の問題については、 コマンドeb eventsと eb health コマンドを使用して詳細情報を取得します。

例

基本的な使用法

インタラクティブモードでの基本的な移行 :

```
PS C:\migrations_workspace >  eb migrate
Identifying VPC configuration of this EC2 instance (i-0123456789abcdef0):
  id: vpc-1234567890abcdef0
```

```
  publicip: true
  elbscheme: public
  ec2subnets: subnet-123,subnet-456,subnet-789
  securitygroups: sg-123,sg-456
  elbsubnets: subnet-123,subnet-456,subnet-789

Using .\migrations\latest to contain artifacts for this migration run.
Generating source bundle for sites, applications, and virtual directories...
  Default Web Site/ -> .\migrations\latest\upload_target\DefaultWebSite.zip

Creating application version
Creating environment

Environment details for: EBMigratedEnv
  Application name: EBMigratedApp
  Region: us-west-2
  Deployed Version: app-230320_153045
  Environment ID: e-abcdef1234
  Platform: 64bit Windows Server 2019 v2.7.0 running IIS 10.0
  Tier: WebServer-Standard-1.0
  CNAME: ebmigratedenv.us-west-2.elasticbeanstalk.com
  Updated: 2023-03-20 15:30:45
```

カスタム設定で特定のサイトを移行する：

```
PS C:\migrations_workspace >  eb migrate `
    --sites "Default Web Site,InternalAPI" `
    --application-name "CorporateApp" `
    --environment-name "Production" `
    --instance-type "c5.xlarge" `
    --tags "Environment=Production,Team=WebOps" `
    --copy-firewall-config
```

デプロイなしで移行アーカイブを作成する：

```
PS C:\migrations_workspace >  eb migrate --archive-only
Using .\migrations\latest to contain artifacts for this migration run.
Generating source bundle for sites, applications, and virtual directories...
  Default Web Site/ -> .\migrations\latest\upload_target\DefaultWebSite.zip

Generated destination archive directory at .\migrations\latest\upload_target
You can execute `eb init` and `eb create` from this directory to deploy to EB.
```

詳細設定の例

JSON ファイルを使用したカスタム VPC 設定による移行：

```
PS C:\migrations_workspace >  cat vpc-config.json
{
    "id": "vpc-1234567890abcdef0",
    "publicip": "false",
    "elbscheme": "internal",
    "ec2subnets": [
        "subnet-private1",
        "subnet-private2"
    ],
    "securitygroups": [
        "sg-app",
        "sg-database",
        "sg-monitoring"
    ],
    "elbsubnets": [
        "subnet-private1",
        "subnet-private2"
    ]
}

PS C:\migrations_workspace eb migrate `
    --sites "InternalAPI" `
    --vpc-config vpc-config.json `
    --instance-type "r5.xlarge" `
    --tags "Environment=Internal,Security=High"
```

SSL 証明書とホストヘッダーを使用したサイトの移行：

```
PS C:\migrations_workspace >  eb migrate `
    --sites "SecurePortal" `
    --ssl-certificates "arn:aws:acm:region:account:certificate/
cert1,arn:aws:acm:region:account:certificate/cert2" `
    --verbose
INFO: Detected HTTPS bindings:
  - www.example.com:443
  - api.example.com:443

INFO: Configuring Application Load Balancer with SSL certificates
INFO: Creating host-based routing rules:
```

```
  - www.example.com -> target group 1
  - api.example.com -> target group 2
```

EBS スナップショット設定を使用した移行:fo

```
PS C:\migrations_workspace >  eb migrate `
    --sites "Default Web Site" `
    --ebs-snapshots "snap-1234567890abcdef0" "snap-0987654321fedcba1" `
    --encrypt-ebs-volumes
Using .\migrations\latest to contain artifacts for this migration run.
INFO: Enabling EBS encryption for all new volumes in us-west-2
INFO: Configuring environment with specified EBS snapshots
```

セキュリティ設定の例

複雑なファイアウォールルールを持つサイトの処理：

```
PS C:\migrations_workspace >  eb migrate `
    --sites "Default Web Site,ReportingService" `
    --copy-firewall-config `
    --verbose
INFO: Detected the following Windows Firewall rules:
  - Allow Web Traffic (TCP 80, 443)
  - Allow Reporting Traffic (TCP 8081)
INFO: Creating corresponding security group rules
```

カスタム IAM ロールを使用した移行：

```
PS C:\migrations_workspace >  eb migrate `
    --sites "SecureApp" `
    --instance-profile "CustomInstanceProfile" `
    --service-role "CustomServiceRole"
```

リモート実行の例

リモート Windows サーバーからの IIS アプリケーションの移行：

```
PS C:\migrations_workspace >  eb migrate `
    --remote `
    --target-ip "192.0.2.10" `
```

```
    --username "administrator" `
    --password "YourPassword123" `
    --application-name "RemoteApp" `
    --environment-name "RemoteEnv"
INFO: Establishing SSH connection to remote host 192.0.2.10...
INFO: Connection established
INFO: Discovering IIS sites on remote host...
INFO: Found 2 sites: Default Web Site, API
INFO: Extracting site configurations...
INFO: Generating source bundle for sites, applications, and virtual directories...
  Default Web Site/ -> .\migrations\latest\upload_target\DefaultWebSite.zip
  API/ -> .\migrations\latest\upload_target\API.zip

Creating application version
Creating environment

Environment details for: RemoteEnv
  Application name: RemoteAppstage mi
  Region: us-west-2
  Deployed Version: app-230320_153045
  Environment ID: e-abcdef1234
  Platform: 64bit Windows Server 2019 v2.7.0 running IIS 10.0
  Tier: WebServer-Standard-1.0
  CNAME: remoteenv.us-west-2.elasticbeanstalk.com
  Updated: 2023-03-20 15:30:45
```

特定のサイト選択によるリモート移行：

```
PS C:\migrations_workspace >  eb migrate `
    --remote `
    --target-ip "192.0.2.10" `
    --username "administrator" `
    --password "YourPassword123" `
    --sites "API" `
    --instance-type "c5.large"
```

eb open

説明

デフォルトのブラウザでウェブサイトのパブリック URL を開きます。

構文

eb open

eb open ***environment-name***

オプション

名前	説明
一般的なオプション	

Output

コマンド eb open では、何も出力されません。代わりに、ブラウザウィンドウでアプリケーションを開きます。

eb platform

説明

このコマンドは、2 つの異なるワークスペースをサポートしています。

プラットフォーム

このワークスペースを使用して、カスタムプラットフォームを管理します。

環境

このワークスペースを使用して、デフォルトのプラットフォームを選択するか、現在のプラットフォームに関する情報を表示します。

Elastic Beanstalk は、eb platform のショートカット ebp を提供します。

> **Note**
>
> Windows PowerShell では、コマンドエイリアスとして ebp を使用します。Windows PowerShell で EB CLI を実行している場合は、このコマンドの長い形式 eb platform を使用してください。

カスタム プラットフォーム用に eb プラットフォームの使用

現在プラットフォームのバージョンを一覧表示し、カスタム プラットフォームを管理することができます。

構文

eb platform create [***version***] [***options***]

eb platform delete [***version***] [***options***]

eb platform events [***version***] [***options***]

eb platform init [***platform***] [***options***]

eb platform list [***options***]

eb platform logs [***version***] [***options***]

eb platform status [***version***] [***options***]

eb platform use [***platform***] [***options***]

オプション

名前	説明
`create [version] [options]`	新しいプラットフォームバージョンを構築します。詳細はこちら。
`delete version [options]`	プラットフォームバージョンを削除します。詳細はこちら。
`events [version] [options]`	プラットフォームバージョンからイベントを表示します。詳細はこちら。
`init [platform] [options]`	プラットフォームリポジトリを初期化します。詳細はこちら。
`list [options]`	現在のプラットフォームのバージョンを一覧表示します。詳細はこちら。

名前	説明
logs [*version*] [*options*]	プラットフォームバージョンのビルダー環境からログを表示します。詳細はこちら。
status [*version*] [*options*]	プラットフォームバージョンのステータスを表示します。詳細はこちら。
use [*platform*] [*options*]	新しいバージョンの構築元の別のプラットフォームを選択します。詳細はこちら。
一般的なオプション	

一般的なオプション

すべての eb platform コマンドは、次の一般的なオプションが含まれます。

名前	説明
-h または --help	ヘルプ メッセージと終了を示します。
--debug	追加デバッグ出力を示します。
--quiet	すべての出力を制限します。
-v または --verbose	追加の出力を示します。
--profile *PROFILE*	認証情報から指定の *PROFILE* を使用します。
-r *REGION* または	リージョン *REGION* を使用してください。

名前	説明
`--region` *REGION*	
`--no-verify-ssl`	SSL AWS 証明書を検証しないでください。

Eb platform create

新しいバージョンのプラットフォームを構築し、新しいバージョンの ARN を返します。現在のリージョンで実行されているビルダー環境がない場合、このコマンドによって環境が起動されます。*#####*と増分オプション (`-M`、`-m`、および `-p`) は、相互に排他的です。

オプション

名前	説明
version	*#####*が指定されていない場合は、パッチバージョン (n.n.N の N) を増分した最新のプラットフォームに基づいて新しいバージョンを作成します。

`-M`
または
`--major-increment`
	メジャーバージョン番号 (n.n.N の N) を増分します。

`-m`
または
`--minor-increment`
	マイナーバージョン番号 (n.n.N の N) を増分します。

`-p`
または
`--patch-increment`
	パッチバージョン番号 (n.n.N の N) を増分します。

`-i` *INSTANCE_TYPE* または	**`t1.micro`** などのインスタンスタイプとして *INSTANCE_TYPE* を使用します。

名前	説明
--instance-type *INSTANCE_TYPE*	
-ip *INSTANCE_PROFILE* または --instance-profile *INSTANCE_PROFILE*	カスタムプラットフォーム用の AMI を作成する場合は、インスタンスプロファイルとして *INSTANCE_PROFILE* を使用します。 -ip オプションを指定しない場合は、インスタンスプロファイル aws-elasticbeanstalk-custom-platforme-ec2-role を作成して、カスタムプラットフォームで使用します。
--tags *key1=value1[,ke*	タグは、key=value ペアのカンマ区切りリストとして指定されます。
--timeout *#*	コマンドがタイムアウトするまでの時間 (分) を設定します。
--vpc.id *VPC_ID*	Packer が構築する VPC の ID。
--vpc.subnets *VPC_SUBNETS*	Packer が構築する VPC のサブネット。
--vpc.publicip	パブリック IP を EC2 インスタンスに関連付けます。

Eb platform delete

プラットフォームバージョンを削除します。環境がそのバージョンを使用している場合、バージョンは削除されません。

オプション

名前	説明
version	削除するバージョン。この値は必須です。
--cleanup	Failed 状態のすべてのプラットフォームバージョンを削除します。

名前	説明
`--all-platforms`	`--cleanup` が指定されている場合は、すべてのプラットフォームの `Failed` 状態のすべてのプラットフォームバージョンを削除します。
`--force`	バージョンを削除するときは確認する必要はありません。

Eb platform イベント

プラットフォームバージョンからイベントを表示します。*#####*を指定した場合は、そのバージョンからイベントを表示します。それ以外の場合は、現在のバージョンからイベントを表示します。

オプション

名前	説明
version	イベントが表示されるバージョン。この値は必須です。

`-f`
または
`--follow`
	発生したイベントを表示し続けます。

Eb platform init

プラットフォームリポジトリを初期化します。

オプション

名前	説明
platform	初期化するプラットフォームの名前。`-i` (インタラクティブモード) が有効になっていない限り、この値は必須です。
`-i` または	インタラクティブモードを使用します。

名前	説明
`--interactive`	
`-k` *`KEYNAME`* または `--keyname` *`KEYNAME`*	デフォルトの EC2 キー名。

以前に初期化したディレクトリで実行した場合は workspace のタイプを変更できませんが、以前に初期化したディレクトリではこのコマンドを実行できます。

異なるオプションで初期化を再初期化するには、オプションを `-i` 使用します。

Eb platform list

ワークスペース (ディレクトリ) またはリージョンに関連付けられるプラットフォームのバージョンを一覧表示します。

次のように、コマンドは実行したワークスペースの種類によって異なる結果を返します。

- プラットフォームワークスペース (`eb platform init` によって初期化されたディレクトリ) では、コマンドはワークスペースで定義されたカスタムプラットフォームのすべてのプラットフォームバージョンのリストを返します。`--all-platforms` または `--verbose` オプションを追加して、アカウントがワークスペースに関連付けられたリージョンにあるすべてのカスタムプラットフォームのすべてのプラットフォームバージョンのリストを取得します。
- アプリケーションワークスペース (`eb init` で初期化されたディレクトリ) では、このコマンドは、Elastic Beanstalk によって管理されているプラットフォームとアカウントのカスタムプラットフォームの両方について、すべてのプラットフォームバージョンの一覧を返します。このリストには短いプラットフォームバージョン名が使用されており、いくつかのプラットフォームバージョンのバリアントを組み合わせることもできます。`--verbose` オプションを追加すると、フルネームとすべてのバリアントが個別にリストされた詳細リストが表示されます。
- 初期化されていないディレクトリでは、コマンドは `--region` オプションでのみ機能します。そのリージョンでサポートされているすべての Elastic Beanstalk マネージドプラットフォームバージョンのリストを返します。このリストには短いプラットフォームバージョン名が使用されており、いくつかのプラットフォームバージョンのバリアントを組み合わせることもできます。`--verbose` オプションを追加すると、フルネームとすべてのバリアントが個別にリストされた詳細リストが表示されます。

オプション

名前	説明

`-a`
または
`--all-platforms`
	初期化されたワークスペース (`eb platform init` または `eb init` によって初期化されたディレクトリ) 内でのみ有効です。アカウントに関連付けられるカスタムプラットフォームすべてのプラットフォームバージョンを一覧表示します。

`-s` *`STATUS`* または `--status` *`STATUS`*	*`STATUS`* と一致するプラットフォームのみを一覧表示します。 • 準備完了 • 失敗 • 削除 • 作成

Eb platform logs

プラットフォームバージョンのビルダー環境からログを表示します。

オプション

名前	説明
`version`	ログが表示されるプラットフォームのバージョン。省略した場合、現行バージョンからログを表示します。
`--stream`	CloudWatch でセットアップされたデプロイログをストリーミングします。

Eb platform status

プラットフォームバージョンのステータスを表示します。

オプション

名前	説明
version	ステータスが取得されるプラットフォームのバージョン。省略すると、現在のバージョンのステータスが表示されます。

Eb platform use

新しいバージョンの構築元の別のプラットフォームを選択します。

オプション

名前	説明
platform	このワークスペースのアクティブなバージョンとして#######を指定します。この値は必須です。

環境に eb プラットフォームを使用

サポートされるプラットフォームを示し、環境を起動するときに使用するデフォルトのプラットフォームとプラットフォームバージョンを設定できるようにします。サポートされるすべてのプラットフォームを一覧表示するには、eb platform list を使用します。プロジェクトのプラットフォームを変更するには、eb platform select を使用します。プロジェクトで選択されたプラットフォームを表示するには、eb platform show を使用します。

構文

eb platform list

eb platform select

eb platform show

オプション

名前	説明
list	現在のプラットフォームのバージョンをリストします。
select	デフォルトプラットフォームを選択します。
show	現在のプラットフォームに関する情報を示します。

例 1

次の例では、Elastic Beanstalk がサポートするすべてのプラットフォームのすべての設定のすべての名前を一覧表示しています。

```
$ eb platform list
docker-1.5.0
glassfish-4.0-java-7-(preconfigured-docker)
glassfish-4.1-java-8-(preconfigured-docker)
go-1.3-(preconfigured-docker)
go-1.4-(preconfigured-docker)
iis-7.5
iis-8
iis-8.5
multi-container-docker-1.3.3-(generic)
node.js
php-5.3
php-5.4
php-5.5
python
python-2.7
python-3.4
python-3.4-(preconfigured-docker)
ruby-1.9.3
ruby-2.0-(passenger-standalone)
ruby-2.0-(puma)
ruby-2.1-(passenger-standalone)
ruby-2.1-(puma)
ruby-2.2-(passenger-standalone)
ruby-2.2-(puma)
tomcat-6
```

```
tomcat-7
tomcat-7-java-6
tomcat-7-java-7
tomcat-8-java-8
```

例 2

次の例では、指定したプラットフォームにデプロイするプラットフォームとバージョンの一覧から選択するよう求めます。

```
$ eb platform select
Select a platform.
1) PHP
2) Node.js
3) IIS
4) Tomcat
5) Python
6) Ruby
7) Docker
8) Multi-container Docker
9) GlassFish
10) Go
(default is 1): 5

Select a platform version.
1) Python 2.7
2) Python
3) Python 3.4 (Preconfigured - Docker)
```

例 3

次の例では、現在のデフォルト プラットフォームに関する情報を示しています。

```
$ eb platform show
Current default platform: Python 2.7
New environments will be running:  64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7

Platform info for environment "tmp-dev":
Current: 64bit Amazon Linux 2014.09 v1.2.0 running Python
Latest:  64bit Amazon Linux 2014.09 v1.2.0 running Python
```

eb printenv

説明

コマンドウィンドウにすべての環境プロパティを出力します。

構文

eb printenv

eb printenv ***environment-name***

オプション

名前	説明
一般的なオプション	

Output

成功すると、コマンドは `printenv` オペレーションのステータスを返します。

例

以下の例では、特定の環境の環境プロパティを出力します。

```
$ eb printenv
Environment Variables:
     PARAM1 = Value1
```

eb restore

説明

終了した環境を再構築し、同じ名前、ID、および設定で新しい環境を作成します。再構築が成功するためには、環境名、ドメイン名、およびアプリケーションバージョンが利用できる必要があります。

構文

eb restore

eb restore ***environment_id***

オプション

名前	説明
一般的なオプション	

Output

EB CLI は、復元可能な終了された環境のリストを表示します。

例

```
$ eb restore
Select a terminated environment to restore

  #   Name          ID             Application Version      Date Terminated        Ago

  3   gamma         e-s7mimej8e9   app-77e3-161213_211138   2016/12/14 20:32 PST   13
 mins
  2   beta          e-sj28uu2wia   app-77e3-161213_211125   2016/12/14 20:32 PST   13
 mins
  1   alpha         e-gia8mphu6q   app-77e3-161213_211109   2016/12/14 16:21 PST   4
 hours

 (Commands: Quit, Restore, # #)

Selected environment alpha
Application:    scorekeep
Description:    Environment created from the EB CLI using "eb create"
CNAME:          alpha.h23tbtbm92.us-east-2.elasticbeanstalk.com
Version:        app-77e3-161213_211109
Platform:       64bit Amazon Linux 2016.03 v2.1.6 running Java 8
Terminated:     2016/12/14 16:21 PST
Restore this environment? [y/n]: y
```

```
2018-07-11 21:04:20    INFO: restoreEnvironment is starting.
2018-07-11 21:04:39    INFO: Created security group named: sg-e2443f72
...
```

eb scale

説明

インスタンスの最小数と最大数を指定の数に設定し、特定の数のインスタンスで常に動作するように環境をスケーリングします。

構文

eb scale ***number-of-instances***

eb scale ***number-of-instances environment-name***

オプション

名前	説明
--timeout	コマンドがタイムアウトするまでの時間 (分) 。
一般的なオプション	

Output

成功すると、コマンドは実行するインスタンスの最小数および最大数を、指定された数に更新します。

例

次の例では、インスタンスの数を 2 に設定します。

```
$ eb scale 2
2018-07-11 21:05:22    INFO: Environment update is starting.
2018-07-11 21:05:27    INFO: Updating environment tmp-dev's configuration settings.
2018-07-11 21:08:53    INFO: Added EC2 instance 'i-5fce3d53' to Auto Scaling Group
 'awseb-e-2cpfjbra9a-stack-AWSEBAutoScalingGroup-7AXY7U13ZQ6E'.
2018-07-11 21:08:58    INFO: Successfully deployed new configuration to environment.
```

```
2018-07-11 21:08:59    INFO: Environment update completed successfully.
```

eb setenv

説明

デフォルトの環境の環境プロパティを設定します。

構文

eb setenv ***key***=***value***

プロパティは必要な数だけ含めることができますが、すべてプロパティの合計サイズが 4,096 バイトを超えることはできません。値を空白にすると、変数を削除できます。制限については、「環境プロパティ (環境変数) の設定」を参照してください。

> **Note**
>
> `value` に特殊文字が含まれている場合は、これらの文字の先頭に \ を付けてエスケープする必要があります。

オプション

名前	説明
--timeout	コマンドがタイムアウトするまでの時間 (分) 。
一般的なオプション	

Output

成功すると、コマンドは環境が正常に更新されたことを表示します。

例

次の例では、環境変数 ExampleVar を設定します。

```
$ eb setenv ExampleVar=ExampleValue
```

```
2018-07-11 21:05:25    INFO: Environment update is starting.
2018-07-11 21:05:29    INFO: Updating environment tmp-dev's configuration settings.
2018-07-11 21:06:50    INFO: Successfully deployed new configuration to environment.
2018-07-11 21:06:51    INFO: Environment update completed successfully.
```

以下のコマンドは、複数の環境プロパティを設定します。このコマンドは、foo という名前の環境プロパティを追加し、その値を bar に設定します。また、JDBC_CONNECTION_STRING プロパティの値を変更し、PARAM4 プロパティと PARAM5 プロパティを削除します。

```
$ eb setenv foo=bar JDBC_CONNECTION_STRING=hello PARAM4= PARAM5=
```

eb ssh

説明

> **Note**
>
> このコマンドは、Windows Server インスタンスを実行する環境では機能しません。

Secure Shell (SSH) を使用して、環境内の Linux Amazon EC2 インスタンスに接続します。環境で複数のインスタンスが実行されている場合、EB CLI では接続するインスタンスを指定するように求められます。このコマンドを使用するには、SSH がローカルマシンにインストールされており、コマンドラインから呼び出せる必要があります。プライベートキーファイルは、ユーザーディレクトリの下の .ssh というフォルダに配置される必要があり、環境の EC2 インスタンスはパブリック IP アドレスを持つ必要があります。

root ディレクトリにカスタムプラットフォームを指定する platform.yaml ファイルが含まれている場合、このコマンドはカスタム環境のインスタンスにも接続します。

> **SSH キー**
>
> 過去に SSH を設定したことがない場合は、EB CLI を使用して、eb init を実行するときにキーを作成できます。すでに eb init を実行している場合は、--interactive オプションを指定して実行し直し、SSH を設定するプロンプトが表示されたら [はい] および [Create New Keypair (新しいキーペアの作成)] を選択します。このプロセスで作成されたキーは、EB CLI によって適切なフォルダに保存されます。

環境のセキュリティグループでポート 22 に関するルールが指定されていない場合、このコマンドは、0.0.0.0/0 (すべての IP アドレス) からの受信トラフィックについてポート 22 を自動的に開きます。セキュリティを高めるために、制限された CIDR 範囲に対してのみポート 22 を開くよう環境のセキュリティグループを設定してある場合は、EB CLI はその設定を尊重し、セキュリティグループに対するあらゆる変更を破棄します。この動作をオーバーライドし、EB CLI ですべての受信トラフィックに対して強制的にポート 22 を開くには、`--force` オプションを使用します。

環境のセキュリティグループの設定については、「EC2 セキュリティグループ」を参照してください。

構文

eb ssh

eb ssh ***environment-name***

オプション

名前	説明

`-i`
または
`--instance`
	接続するインスタンスのインスタンス ID を指定します。このオプションを使用することをお勧めします。

`-n`
または
`--number`
	接続するインスタンスを数値で指定します。

`-o`
または
`--keep_open`
	SSH セッションの終了後、セキュリティグループでポート 22 を開いたままにします。

`--command`	SSH セッションを開始する代わりに、指定されたインスタンスでシェルコマンドを実行します。

名前	説明
`--custom`	'ssh -i keyfile' の代わりに使用する SSH コマンドを指定します。リモートユーザーとホスト名を含めないでください。
`--setup`	環境のインスタンスに割り当てられているキーペアを変更します (インスタンスを置き換える必要があります)。
`--force`	環境のセキュリティグループ SSH に関する設定がすでに行われている場合でも、0.0.0.0/0 からの受信トラフィックに対してポート 22 を開きます。 接続しようとしている IP アドレスが含まれない、制限された CIDR 範囲に対してポート 22 を開くように環境のセキュリティグループが設定されている場合に、このオプションを使用します。
`--timeout #`	コマンドがタイムアウトするまでの時間 (分) を設定します。 `--setup` 引数にのみ使用できます。
一般的なオプション	

Output

成功すると、コマンドはインスタンスへの SSH 接続を開きます。

例

次の例では、指定した環境に接続します。

```
$ eb ssh
Select an instance to ssh into
1) i-96133799
2) i-5931e053
(default is 1): 1
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
The authenticity of host '54.191.45.125 (54.191.45.125)' can't be established.
RSA key fingerprint is ee:69:62:df:90:f7:63:af:52:7c:80:60:1b:3b:51:a9.
```

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.191.45.125' (RSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2014.09-release-notes/
No packages needed for security; 1 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-8-185 ~]$ ls
[ec2-user@ip-172-31-8-185 ~]$ exit
logout
Connection to 54.191.45.125 closed.
INFO: Closed port 22 on ec2 instance security group
```

eb status

説明

環境のステータスに関する情報が提供されます。

root ディレクトリにカスタムプラットフォームを指定する `platform.yaml` ファイルが含まれている場合、このコマンドはビルダー環境に関する情報も表示します。

構文

eb status

eb status ***environment-name***

オプション

名前	説明

`-v`
または
`--verbose`
	個々のインスタンスについてより詳細な情報（Elastic Load Balancing ロードバランサーに関するインスタンスのステータスなど）を提供します。

名前	説明
一般的なオプション	

Output

成功すると、コマンドは環境に関する次の情報を返します。

- 環境名
- アプリケーション名
- デプロイされたアプリケーションバージョン
- 環境 ID
- プラットフォーム
- 環境枠
- CNAME
- 環境が最後に更新された日時
- ステータス
- ヘルス

詳細モードを使用すると、EB CLI は、実行中の Amazon EC2 インスタンスの数も表示します。

例

次の例では、環境 tmp-dev のステータスを示しています。

```
$ eb status
Environment details for: tmp-dev
  Application name: tmp
  Region: us-west-2
  Deployed Version: None
  Environment ID: e-2cpfjbra9a
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
  Updated: 2014-10-29 21:37:19.050000+00:00
  Status: Launching
```

```
  Health: Grey
```

eb swap

説明

環境の CNAME を別の環境の CNAME と交換します (たとえば、アプリケーションのバージョンを更新する際にダウンタイムを回避するため)。

> **Note**
>
> 2 つ以上の環境がある場合、希望する CNAME を現在使用している環境名を環境リストから選択するよう求められます。これを制限するには、コマンドを実行する際に -n オプションで使用する環境の名前を指定します。

構文

eb swap

eb swap ***environment-name***

> **Note**
>
> *environment-name* は、異なる CNAME を使用する環境です。eb swap を実行する際にコマンドラインパラメータに *environment-name* を指定しないと、EB CLI はデフォルトの環境の CNAME を更新します。

オプション

名前	説明
-n または --destination_name	CNAME を交換したい環境の名前を指定します。このオプションを指定しないで eb swap を実行すると、EB CLI は環境のリストから選択するよう求められます。

名前	説明
一般的なオプション	

Output

成功すると、コマンドは swap オペレーションのステータスを返します。

例

以下の例は環境 tmp-dev を live-env と交換します。

```
$ eb swap
Select an environment to swap with.
1) staging-dev
2) live-env
(default is 1): 2
2018-07-11 21:05:25    INFO: swapEnvironmentCNAMEs is starting.
2018-07-11 21:05:26    INFO: Swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
2018-07-11 21:05:30    INFO: 'tmp-dev.elasticbeanstalk.com' now points to 'awseb-e-j-
AWSEBLoa-M7U21VXNLWHN-487871449.us-west-2.elb.amazonaws.com'.
2018-07-11 21:05:30    INFO: Completed swapping CNAMEs for environments 'tmp-dev' and
 'live-env'.
```

以下の例は環境 tmp-dev を環境 live-env と交換しますが、設定に値の入力または選択を求めません。

```
$ eb swap tmp-dev --destination_name live-env
2018-07-11 21:18:12    INFO: swapEnvironmentCNAMEs is starting.
2018-07-11 21:18:13    INFO: Swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
2018-07-11 21:18:17    INFO: 'tmp-dev.elasticbeanstalk.com' now points to 'awseb-e-j-
AWSEBLoa-M7U21VXNLWHN-487871449.us-west-2.elb.amazonaws.com'.
2018-07-11 21:18:17    INFO: Completed swapping CNAMEs for environments 'tmp-dev' and
 'live-env'.
```

eb tags

説明

Elastic Beanstalk リソースのタグを追加、削除、更新、一覧表示します。

Elastic Beanstalk でのリソースへのタグ付けの詳細については、「Elastic Beanstalk アプリケーションリソースのタグ付け」を参照してください。

構文

eb tags [*environment-name*] [--resource *ARN*] -l | --list

eb tags [*environment-name*] [--resource *ARN*] -a | --add *key1*=*value1*[,*key2*=*value2* ...]

eb tags [*environment-name*] [--resource *ARN*] -u | --update *key1*=*value1*[,*key2*=*value2* ...]

eb tags [*environment-name*] [--resource *ARN*] -d | --delete *key1*[,*key2* ...]

`--add`、`--update`、`--delete` のサブコマンドオプションは、1 つのコマンドで組み合わせることができます。少なくとも 1 つのサブコマンドオプションが必要です。これらの 3 つのサブコマンドオプションを `--list` で組み合わせることはできません。

追加の引数を指定しない場合、これらすべてのコマンドは、現在のディレクトリのアプリケーションでデフォルト環境のタグを一覧表示または変更します。*`environment-name`* 引数を使用すると、コマンドはその環境のタグを一覧表示または変更します。`--resource` オプションを使用すると、コマンドは任意の Elastic Beanstalk リソース (アプリケーション、環境、アプリケーションバージョン、保存された設定、またはカスタムプラットフォームバージョン) のタグを一覧表示または変更します。リソースを Amazon リソースネーム (ARN) で指定します。

オプション

これらのオプションはいずれも必須ではありません。オプションを使用せずに eb create を実行すると、各設定の値を入力または選択するよう求められます。

名前	説明

`-l`
または
`--list`
	リソースに現在適用されているすべてのタグを一覧表示します。

`-a` *`key1`*`=`*`value1`*`[,`*`key2`*`=`*`value2`* または	リソースに新しいタグを適用します。`key=value` ペアのカンマ区切りリストとしてタグを指定します。既存のタグのキーを指定することはできません。

名前	説明
`--add key1=value1[,key2=val`	有効な値: 「リソースのタグ付け」を参照。

`-u key1=value1[,key2=value2`
 	
または
 	
`--update key1=value1[,key2=value2 .`
	既存のリソースタグの値を更新します。key=value ペアのカンマ区切りリストとしてタグを指定します。既存のタグのキーを指定する必要があります。
	
	有効な値: 「リソースのタグ付け」を参照。

`-d key1[,key2 ...]`
 	
または
 	
`--delete key1[,key2 ...]`
	既存のリソースタグを削除します。キーのカンマ区切りリストとしてタグを指定します。既存のタグのキーを指定する必要があります。
	
	有効な値: 「リソースのタグ付け」を参照。

`-r region`
 	
または
 	
`--region region`
	AWS リージョン リソースが存在する 。
	
	デフォルト: 設定されたデフォルトリージョン。
	
	このオプションに指定できる値のリストについては、「AWS 全般のリファレンス」の「AWS Elastic Beanstalk エンドポイントとクォータ」を参照してください。

`--resource ARN`	コマンドでタグを変更または一覧表示するリソースの ARN。指定しない場合、コマンドは現在のディレクトリのアプリケーションにおける (デフォルトまたは指定された) 環境を参照します。 有効な値については、該当するリソース別の リソースのタグ付け のサブトピックを参照してください。これらのトピックでは、リソースの ARN の構築方法を示し、該当するリソースの ARN のリストをアプリケーションまたはアカウントで取得する方法について説明しています。

Output

`--list` サブコマンドオプションは、リソースのタグを一覧表示します。出力には、Elastic Beanstalk でデフォルトで適用するタグとカスタムタグの両方が表示されます。

```
$ eb tags --list
Showing tags for environment 'MyApp-env':

Key                                 Value

Name                                MyApp-env
elasticbeanstalk:environment-id     e-63cmxwjaut
elasticbeanstalk:environment-name   MyApp-env
mytag                               tagvalue
tag2                                2nd value
```

成功した場合、`--add`、`--update`、`--delete` のサブコマンドオプションには出力されません。コマンドのアクティビティの詳細な出力を表示するには、`--verbose` オプションを追加します。

```
$ eb tags --verbose --update "mytag=tag value"
Updated Tags:

Key                                 Value

mytag                               tag value
```

例

次のコマンドは、キー `tag1` と値 `value1` を使用するタグをアプリケーションのデフォルト環境に正常に追加し、同時にタグ `tag2` を削除します。

```
$ eb tags --add tag1=value1 --delete tag2
```

次のコマンドは、アプリケーション内の保存された設定にタグを正常に追加します。

```
$ eb tags --add tag1=value1 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-
id:configurationtemplate/my-app/my-template"
```

次のコマンドは、存在しないタグを更新しようとしているため失敗します。

```
$ eb tags --update tag3=newval
ERROR: Tags with the following keys can't be updated because they don't exist:

  tag3
```

次のコマンドは、同じキーを更新しようとしているため失敗します。

```
$ eb tags --update mytag=newval --delete mytag
ERROR: A tag with the key 'mytag' is specified for both '--delete' and '--update'. Each
 tag can be either deleted or updated in a single operation.
```

eb terminate

説明

実行中の環境を終了して、未使用の AWS リソースに対して料金が発生しないようにします。

[`--all`] オプションを使用して eb init を使って、現在のディレクトリが初期化されたアプリケーションを削除します。このコマンドは、アプリケーション内のすべての環境を終了します。さらに、アプリケーションのアプリケーションバージョンと保存された設定を終了し、アプリケーションを削除します。

root ディレクトリにカスタムプラットフォームを指定する `platform.yaml` ファイルが含まれている場合、このコマンドは実行中のカスタム環境を終了します。

> **Note**
>
> いつでも、また同じバージョンを使用して新しい環境を起動できます。

保存する必要のある環境のデータがある場合、環境を終了する前にデータベース削除ポリシーを `Retain` に設定します。これにより、データベースは Elastic Beanstalk の外部で動作し続けます。この後、Elastic Beanstalk 環境は外部データベースとして接続する必要があります。データベースを動作させずにデータをバックアップする場合は、環境を終了する前にデータベースのスナップショットを作成するように削除ポリシーを設定します。詳細については、このガイドの環境の設定の章で「データベースのライフサイクル」を参照してください。

> **Important**
>
> 環境を終了する場合は、作成した CNAME マッピングも削除する必要があります。これにより、使用可能になったホスト名を他のお客様が再利用できます。DNS エントリのダングリングを防ぐため、終了した環境を指す DNS レコードを必ず削除してください。DNS エントリがダングリングしていると、ユーザーのドメイン宛のインターネットトラフィックがセキュリティの脆弱性にさらされる可能性があります。また、他のリスクをもたらす可能性もあります。
> 詳細については、Amazon Route 53 デベロッパーガイドの「Route 53 でのダングリング委任レコードからの保護」を参照してください。また、ダングリング DNS エントリの詳細については、AWS セキュリティブログの 「Enhanced Domain Protections for Amazon CloudFront Requests」を参照してください。

構文

eb terminate

eb terminate ***environment-name***

オプション

名前	説明
`--all`	アプリケーションのすべての環境を終了させて、アプリケーションのアプリケーションバージョンを終了し、設定を保存した後で、アプリケーションを削除します。
`--force`	確認のプロンプトを表示しないで環境を終了します。
`--ignore-links`	リンクされた依存する環境がある場合でも、環境を終了します。環境を構成する を参照してください。
`--timeout`	コマンドがタイムアウトするまでの時間 (分) 。

Output

成功すると、コマンドは `terminate` オペレーションのステータスを返します。

例

次のサンプルリクエストでは、環境 tmp-dev を終了します。

```
$ eb terminate
The environment "tmp-dev" and all associated instances will be terminated.
To confirm, type the environment name: tmp-dev
2018-07-11 21:05:25    INFO: terminateEnvironment is starting.
2018-07-11 21:05:40    INFO: Deleted CloudWatch alarm named: awseb-e-2cpfjbra9a-stack-
AWSEBCloudwatchAlarmHigh-16V08YOF2KQ7U
2018-07-11 21:05:41    INFO: Deleted CloudWatch alarm named: awseb-e-2cpfjbra9a-stack-
AWSEBCloudwatchAlarmLow-6ZAWH9F20P7C
2018-07-11 21:06:42    INFO: Deleted Auto Scaling group policy named:
 arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:5d7d3e6b-
d59b-47c5-b102-3e11fe3047be:autoScalingGroupName/awseb-e-2cpfjbra9a-stack-
AWSEBAutoScalingGroup-7AXY7U13ZQ6E:policyName/awseb-e-2cpfjbra9a-stack-AWSEBAutoSca
lingScaleUpPolicy-1876U27JEC34J
2018-07-11 21:06:43    INFO: Deleted Auto Scaling group policy named:
 arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:29c6e7c7-7ac8-46fc-91f5-
cfabb65b985b:autoScalingGroupName/awseb-e-2cpfjbra9a-stack-
AWSEBAutoScalingGroup-7AXY7U13ZQ6E:policyName/awseb-e-2cpfjbra9a-stack-AWSEBAutoSca
lingScaleDownPolicy-SL4LHODMOMU
2018-07-11 21:06:48    INFO: Waiting for EC2 instances to terminate. This may take a
 few minutes.
2018-07-11 21:08:55    INFO: Deleted Auto Scaling group named: awseb-e-2cpfjbra9a-
stack-AWSEBAutoScalingGroup-7AXY7U13ZQ6E
2018-07-11 21:09:10    INFO: Deleted security group named: awseb-e-2cpfjbra9a-stack-
AWSEBSecurityGroup-XT4YYGFL7I99
2018-07-11 21:09:40    INFO: Deleted load balancer named: awseb-e-2-AWSEBLoa-
AK6RRYFQVV3S
2018-07-11 21:09:42    INFO: Deleting SNS topic for environment tmp-dev.
2018-07-11 21:09:52    INFO: terminateEnvironment completed successfully.
```

eb upgrade

説明

現在実行中のプラットフォームの最新バージョンに環境のプラットフォームをアップグレードします。

root ディレクトリにカスタムプラットフォームを指定する `platform.yaml` ファイルが含まれている場合、このコマンドは、現在実行しているカスタムプラットフォームの最新バージョンに環境をアップグレードします。

構文

eb upgrade

eb upgrade ***environment-name***

オプション

名前	説明
`--force`	アップグレード処理を開始する前に環境名の確認を必要とすることなく、アップグレードを実行します。
`--noroll`	アップグレード中に実行中の一部のインスタンスを維持するため、ローリング更新を使用せずにすべてのインスタンスを更新します。
一般的なオプション	

Output

このコマンドは変更の概要を示し、環境名を入力することにより、アップグレードの確認を求めます。成功すると、環境はアップグレードされてから、プラットフォームの最新バージョンで起動されます。

例

次の例では、指定した環境の現在プラットフォームバージョンを使用可能な最新のプラットフォームバージョンにアップグレードします。

```
$ eb upgrade
Current platform: 64bit Amazon Linux 2014.09 v1.0.9 running Python 2.7
Latest platform:  64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7
```

```
WARNING: This operation replaces your instances with minimal or zero downtime. You may
 cancel the upgrade after it has started by typing "eb abort".
You can also change your platform version by typing "eb clone" and then "eb swap".

To continue, type the environment name:
```

eb use

説明

指定した環境をデフォルトの環境として設定します。

Git を使用するときに、eb use は現在のブランチのデフォルト環境を設定します。Elastic Beanstalk にデプロイする各ブランチで、このコマンドを 1 回実行します。

構文

eb use ***environment-name***

オプション

名前	説明
`--source codecommit/` *`repository-name`*`/`*`branch-name`	CodeCommit リポジトリとブランチ。
`-r` *`region`* `--region` *`region`	環境を作成するリージョンを変更します。
一般的なオプション	

Elastic Beanstalk の概念を理解する

概念と用語を理解しておくと、Elastic Beanstalk でアプリケーションをデプロイするために必要な理解が得られます。

アプリケーション

Elastic Beanstalk アプリケーションは、環境、バージョン、環境設定など、Elastic Beanstalk コンポーネントのコンテナです。Elastic Beanstalk アプリケーション内では、コードの実行に関連するすべてのリソースを管理します。

アプリケーションバージョン

Elastic Beanstalk では、アプリケーションバージョンとは、ウェブアプリケーションのデプロイ可能コードの特定のラベル付きイテレーションのことです。アプリケーションバージョンは、Java の WAR ファイルなどのデプロイ可能コードが含まれている Amazon Simple Storage Service（Amazon S3）オブジェクトを指します。

アプリケーションバージョンはアプリケーションの一部です。アプリケーションは多数のバージョンを持つことができ、各アプリケーションバージョンは一意です。実行中の環境では、アプリケーションに既にアップロードしてあるアプリケーションバージョンをデプロイしたり、新しいアプリケーションバージョンをアップロードしてすぐにデプロイしたりできます。たとえば、複数のアプリケーションバージョンをアップロードして、それらの違いをテストできます。

環境

環境は、アプリケーションバージョンを実行する AWS リソースのコレクションです。各環境が実行するのは一度に 1 つのアプリケーションバージョンだけですが、同じアプリケーションバージョンや複数の異なるアプリケーションバージョンを多数の環境で同時に実行できます。環境を作成すると、Elastic Beanstalk は、指定したアプリケーションバージョンを実行するために必要なリソースを AWS アカウントでプロビジョニングします。

環境枠

Elastic Beanstalk 環境を起動したら、まず環境枠を選択します。環境階層は、環境が実行するアプリケーションのタイプを指定し、Elastic Beanstalk がそれをサポートするためにプロビジョニングする

リソースを決定します。HTTP リクエストを処理するアプリケーションは、ウェブサーバー環境枠で実行されます。Amazon Simple Queue Service (Amazon SQS) キューからタスクを取り出すバックエンド環境は、ワーカー環境枠で実行されます。

環境設定

環境設定は、環境とその環境に関連付けられているリソースの動作を定義するパラメータと設定のコレクションを識別します。環境の設定を更新すると、(変更の種類に応じて) Elastic Beanstalk が自動的に既存のリソースを変更または削除し、新しいリソースをデプロイします。

保存された設定

保存された設定 は、一意の環境設定を作成するための開始点として使用できるテンプレートです。保存済み設定を作成および変更し、Elastic Beanstalk コンソール、EB CLI、 AWS CLIまたは API を使用して環境に適用できます。API と は、保存された設定を設定テンプレートとして AWS CLI 参照します。

プラットフォーム

プラットフォームは、オペレーティングシステム、プログラミング言語ランタイム、ウェブサーバー、アプリケーションサーバー、および Elastic Beanstalk コンポーネントの組み合わせです。ウェブアプリケーションを設計し、プラットフォームをターゲットとします。Elastic Beanstalk は、アプリケーションを構築できるさまざまなプラットフォームを提供します。

詳細については、Elastic Beanstalk プラットフォーム を参照してください

Elastic Beanstalk ウェブサーバー環境

次の図は、ウェブサーバー環境枠の Elastic Beanstalk アーキテクチャの例と、そのタイプの環境枠でのコンポーネントの連携を示しています。

環境はアプリケーションの中心です。この図では、環境は最上位の実線内に示されています。環境を作成すると、Elastic Beanstalk はアプリケーションの実行に必要なリソースをプロビジョニングします。環境用に作成された AWS リソースには、1 つの Elastic Load Balancer (図の ELB)、Auto Scaling グループ、および 1 つ以上の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスが含まれます。

すべての環境に、ロードバランサーを指定する CNAME (URL) が含まれます。環境には、`myapp.us-west-2.elasticbeanstalk.com` などの URL があります。この URL は、Amazon Route 53 で CNAME レコードを使用することによって、`abcdef-123456.us-west-2.elb.amazonaws.com` のような別名の Elastic Load Balancing URL になります。Amazon Route 53 は、可用性と拡張性に優れたドメインネームシステム (DNS) ウェブサービスです。このサービスは、インフラストラクチャに対して安全で信頼できるルーティングを提供します。DNS プロバイダに登録したドメイン名は、CNAME にリクエストを転送します。

ロードバランサーは、Auto Scaling グループに属する Amazon EC2 インスタンスの前に配置されています Amazon EC2 Auto Scaling は、アプリケーションへの負荷の増大に対応するために追加の Amazon EC2 インスタンスを自動的に開始します。アプリケーションへの負荷が軽減されると、Amazon EC2 Auto Scaling はインスタンスを停止しますが、少なくとも 1 つのインスタンスは実行されたままです。

Amazon EC2 インスタンスで実行するソフトウェアスタックは、コンテナタイプに応じて変わります。コンテナの種類によって、その環境に使用するインフラストラクチャのトポロジとソフトウェアスタックが定義されます。例えば、Apache Tomcat コンテナを含む Elastic Beanstalk 環境は、Amazon Linux オペレーティングシステム、Apache ウェブサーバー、および Apache Tomcat ソフトウェアを使用します。サポートされているコンテナタイプのリストについては、「Elastic Beanstalk でサポートされているプラットフォーム」を参照してください。これらのコンテナタイプのいずれかが、アプリケーションを実行する各 Amazon EC2 インスタンスによって使用されます。さらに、各 Amazon EC2 インスタンスでは、ホストマネージャー (HM) と呼ばれるソフトウェアコンポーネントも実行されます ホストマネージャーは以下を行います。

- アプリケーションのデプロイ
- イベントとメトリクスの収集と取得 (コンソール、API、またはコマンドラインを使用) 。
- インスタンスレベルのイベントの生成
- アプリケーションログファイルで重大エラーがないかどうかを監視
- アプリケーションサーバーの監視
- インスタンスコンポーネントへの修正プログラムの適用
- アプリケーションのログファイルのローテーションと、Amazon S3 への公開

ホストマネージャーは、メトリクス、エラーとイベント、サーバーインスタンスのステータスをレポートします。これらの情報は、Elastic Beanstalk コンソール、API、CLI を通じて利用できます。

この図では、Amazon EC2 インスタンスが 1 つのセキュリティグループに属しています。セキュリティグループは、インスタンスのファイアウォールのルールを定義します。デフォルトでは、Elastic

Beanstalk によって 1 つのセキュリティグループが定義されます。この場合、全員がポート 80 (HTTP) を使用して接続できます。複数のセキュリティグループを定義することもできます。たとえば、データベースサーバーに対してセキュリティグループを定義できます。Amazon EC2 セキュリティグループの詳細、および Elastic Beanstalk アプリケーションに対してこのセキュリティグループを設定する方法については、「EC2 セキュリティグループ」を参照してください。

Elastic Beanstalk ワーカー環境

AWS ワーカー環境枠用に作成された リソースには、Auto Scaling グループ、1 つ以上の Amazon EC2 インスタンス、IAM ロールが含まれます。また、ワーカー環境枠で Amazon SQS キューがない場合、Elastic Beanstalk によって Amazon SQS キューが作成され、プロビジョニングされます。ワーカー環境を起動すると、選択したプログラミング言語に必要なサポートファイルとデーモンが、Elastic Beanstalk によって Auto Scaling グループの各 EC2 インスタンスにインストールされます。デーモンは Amazon SQS キューからのメッセージを読み取ります。デーモンは、読み取る各メッセージから処理のためにワーカーアプリケーションを実行しているウェブアプリケーションにデータを送信します。ワーカー環境に複数のインスタンスがある場合、各インスタンスに独自のデーモンがありますが、読み取りはすべて同じ Amazon SQS キューからです。

次の図は、さまざまなコンポーネントと、環境と AWS サービス間の相互作用を示しています。

Amazon CloudWatch は、アラームとヘルスモニタリングに使用されます。詳細については、「ベーシックヘルスレポート」を参照してください。

ワーカー環境枠の仕組みの詳細については、「Elastic Beanstalk ワーカー環境」を参照してください。

Elastic Beanstalk アプリケーションの設計上の考慮事項

を使用してデプロイされたアプリケーションは AWS クラウド リソースで AWS Elastic Beanstalk 実行されるため、アプリケーションを最適化するために、スケーラビリティ、セキュリティ、永続的ストレージ、耐障害性、コンテンツ配信、ソフトウェアの更新とパッチ適用、接続など、いくつかの設定要素に注意する必要があります。このトピックでは、これらのそれぞれについて個別に説明します。アーキテクチャやセキュリティ、経済などのトピックをカバーする技術 AWS ホワイトペーパーの包括的なリストについては、AWS 「クラウドコンピューティングホワイトペーパー」を参照してください。

スケーラビリティ

物理ハードウェア環境で作業する場合は、クラウド環境とは対照的に、2 つの方法のいずれかでスケーラビリティにアプローチできます。垂直スケーリングでスケールアップすることも、水平スケーリングを使用してスケールアウトすることもできます。スケールアップアプローチでは、ビジネスの需要の高まりをサポートできる強力なハードウェアに投資する必要があります。スケールアウトアプローチでは、分散型の投資モデルに従う必要があります。そのため、ハードウェアとアプリケーションの取得でよりターゲットを絞れるようになり、データセットをフェデレーションしたり、設計をサービス指向にすることができます。スケールアップアプローチは非常に高くつく可能性があるにもかかわらず、需要が容量を超えてしまうというリスクは依然として存在します。この点で、スケールアウトアプローチは通常、より効果的です。ただし、これを使用する場合は、一定の間隔で需要を予測し、その需要を満たすようにインフラストラクチャをチャンクに展開できる必要があります。その結果、このアプローチでは余剰容量が発生することがよくあるため、注意深くモニタリングする必要があります。

クラウドに移行することで、クラウドの伸縮性を利用して、需要に合わせてインフラストラクチャを作成できます。伸縮性は、リソースの取得とリリースを合理化するのに役立ちます。これにより、需要の増減に合わせてインフラストラクチャを迅速にスケールイン/スケールアウトできるようになります。これを使用するには、環境内のリソースからのメトリクスに基づいてスケールアップまたはスケールダウンするように、Auto Scaling 設定を行います。たとえば、サーバー使用率やネットワーク I/O などのメトリクスを設定できます。Auto Scaling を使用すると、使用量が増加するたびにコンピューティングキャパシティが自動的に追加され、使用量が減少するたびに削除できます。システムメトリクス (CPU、メモリ、ディスク I/O、ネットワーク I/O など) を Amazon CloudWatch に発行できます。次に、CloudWatch を使用して Auto Scaling アクションをトリガーしたり、これらのメトリクスに基づいて通知を送信したりするアラームを設定できます。Auto Scaling を設定する方法については、Elastic Beanstalk 環境インスタンスの Auto Scaling を参照してください。

また、必要に応じてスケールアウトすることができる耐障害性に優れた疎結合コンポーネントを使用して、すべての Elastic Beanstalk アプリケーションを可能な限りステートレスに設計することをお勧めします。のスケーラブルなアプリケーションアーキテクチャの設計の詳細についてはAWS、AWS 「Well-Architected Framework」を参照してください。

セキュリティ

のセキュリティ AWS は責任共有です。Amazon Web Services は、お客様の環境の物理リソースを保護し、クラウドがお客様のアプリケーションを実行するための安全な場所であることを保証します。お客様の Elastic Beanstalk 環境で送受信されるデータのセキュリティおよびお客様のアプリケーションのセキュリティに対しては、お客様に責任があります。

アプリケーションとクライアントの間で流れる情報を保護するため、SSL を設定します。SSL を設定するには、 AWS Certificate Manager (ACM) からの無料の証明書が必要です。外部証明機関 (CA) からの証明書がすでにある場合は、ACM を使用し、ACM を使用してその証明書をインポートできます。それ以外の場合は、 を使用してインポートできます AWS CLI。

ACM が で利用 AWS リージョンできない場合は、VeriSign や™ などの外部 CA から証明書を購入できます。次に、 AWS Command Line Interface （ AWS CLI) を使用して、サードパーティーまたは自己署名証明書とプライベートキーを AWS Identity and Access Management (IAM) にアップロードします。証明書の公開キーにより、ブラウザに対してサーバーが認証されます。また、公開キーは、双方向のデータを暗号化する共有セッションキーを作成するための基盤となります。SSL 証明書の作成、アップロード、および環境に割り当てる方法については、Elastic Beanstalk 環境の HTTPS の設定 を参照してください。

環境に SSL 証明書を設定する場合、クライアントとお客様の環境の Elastic Load Balancing ロードバランサーの間でデータが暗号化されます。デフォルトでは、暗号化はロードバランサーで終了し、ロードバランサーと Amazon EC2 インスタンス間のトラフィックは暗号化されません。

永続ストレージ

Elastic Beanstalk アプリケーションは、永続的ローカルストレージがない Amazon EC2 インスタンスで実行されます。Amazon EC2 インスタンスが終了した場合、ローカルファイルシステムは保存されません。新しい Amazon EC2 インスタンスは、デフォルトのファイルシステムから始まります。永続的データソースにデータを保存するようアプリケーションを設定することをお勧めします。AWS には、アプリケーションに使用できる複数の永続的ストレージサービスが用意されています。以下の表にそれらの設定を示します。

ストレージサービス	サービスのドキュメント	Elastic Beanstalk の統合
Amazon S3	Amazon Simple ストレージ Service ドキュメント	Amazon S3 で Elastic Beanstalk を使用する
Amazon Elastic File System	Amazon Elastic File System ドキュメント	Amazon Elastic File System で Elastic Beanstalk を使用する
Amazon Elastic Block Store	Amazon Elastic Block Store 特徴ガイド: Elastic Block Store	

ストレージサービス	サービスのドキュメント	Elastic Beanstalk の統合
Amazon DynamoDB	Amazon DynamoDB ドキュメント	Amazon DynamoDB で Elastic Beanstalk を使用する
Amazon Relational Database Service (RDS)	Amazon Relational Database Service ドキュメント	Amazon RDS で Elastic Beanstalk を使用する

> **Note**
>
> Elastic Beanstalk は EC2 インスタンス上のアプリケーションディレクトリの所有者としてセットアップするための webapp ユーザーを作成します。2022年2月3日 以降にリリースされた Amazon Linux 2 プラットフォームバージョンの場合、Elastic Beanstalk は、新しい環境の webapp ユーザーに uid (ユーザー ID) と gid (グループ ID) の値 900 を割り当てます。これは、プラットフォームバージョンの更新後の既存の環境でも同じです。この方法により、webapp ユーザーは、永続的なファイルシステムストレージに一貫してアクセスできるようになります。
> 別のユーザーまたはプロセスがすでに 900 を使用しているという可能性は低い状況では、オペレーティングシステムがデフォルトで webapp ユーザー uid と gid を別の値に設定します。Linux コマンドを実行する id webapp EC2 インスタンスで、webapp ユーザーに割り当てられている uid および gid 値を検証します。。

耐障害性

簡単に言うと、クラウドでのアーキテクチャ設計は、悪いことを想定しながら行うことをお勧めします。それが提供する伸縮性を活用。いつも故障から自動回復できるように設計、実行、およびデプロイしてください。Amazon EC2 インスタンスおよび Amazon RDS については、複数のアベイラビリティーゾーンを使用します。アベイラビリティーゾーンは概念的には論理データセンターに似ています。Amazon CloudWatch は、Elastic Beanstalk アプリケーションのヘルスを詳しく確認するときに使用してください。これにより、ハードウェア障害やパフォーマンス低下が発生した場合に適切な措置を講じることができます。問題のある Amazon EC2 インスタンスを新しいインスタンスに置き換えられるように、一連の Amazon EC2 インスタンスのサイズを一定に保ちながら維持するには、Auto Scaling を設定します。Amazon RDS を使用してバックアップの保存期間を設定すると、Amazon RDS によって自動バックアップを実行できます。

コンテンツ配信

ユーザーがウェブサイトに接続している場合、そのユーザーのリクエストは複数の個別のネットワークを介してルーティングされる可能性があります。その結果、レイテンシーが増え、パフォーマンスが低下することがあります。Amazon CloudFront は、世界各地に設置されたエッジロケーションのネットワークを介して画像や動画などウェブコンテンツを配信することで、こうしたレイテンシーの問題を改善します。ユーザーのリクエストがルーティングされるのは最寄りのエッジロケーションです。したがって、コンテンツは可能な限り最良のパフォーマンスで配信されます。CloudFront は、ファイルの元の最終バージョンをしっかりと保存する、Amazon S3 とシームレスに連携します。Amazon CloudFront の使用に関する詳細については、Amazon CloudFront デベロッパーガイドを参照してください。

ソフトウェアの更新プログラムと修正プログラム

AWS Elastic Beanstalk は、プラットフォームの更新を定期的にリリースして、修正、ソフトウェアの更新、新機能を提供します。Elastic Beanstalk は、プラットフォームの更新を処理するためのいくつかのオプションを提供しています。マネージドプラットフォーム更新機能により、アプリケーションが稼働している間、予定済みのメンテナンスウィンドウ中に、環境を自動的に最新バージョンのプラットフォームにアップグレードできます。Elastic Beanstalk コンソールを使用して 2019 年 11 月 25 日以降に作成された環境では、マネージドアップデートは可能な限りデフォルトで有効になります。Elastic Beanstalk コンソールまたは EB CLI を使用して、手動で更新を開始することもできます。

接続

デプロイを完了するには、環境内のインスタンスに Elastic Beanstalk が接続できる必要があります。Elastic Beanstalk アプリケーションを Amazon VPC 内にデプロイする場合、接続を有効にするために必要な設定は、作成する Amazon VPC 環境のタイプによって異なります。

- 単一インスタンス環境では、追加の設定は必要ありません。これは、これらの環境では、Elastic Beanstalk によって各 Amazon EC2 インスタンスにパブリック Elastic IP アドレスが割り当てられ、これによりインスタンスが直接インターネットと通信できるようになるためです。
- パブリックサブネットとプライベートサブネットの両方を備えた Amazon VPC 内の負荷分散されたスケーラブルな環境の場合、次の作業を行う必要があります。
  - インターネットから Amazon EC2 インスタンスへのインバウンドトラフィックをルーティングするロードバランサーをパブリックサブネットに作成します。
  - ネットワークアドレス変換 (NAT) デバイスを作成して、プライベートサブネットの Amazon EC2 インスタンスからインターネットにアウトバウンドトラフィックをルーティングします。

  - プライベートサブネット内の Amazon EC2 インスタンスのインバウンドおよびアウトバウンドルーティングルールを作成します。
  - NAT インスタンスを使用している場合は、NAT インスタンスと Amazon EC2 インスタンスのセキュリティグループを設定して、インターネット通信を有効にします。
- Amazon VPC 内の負荷分散されたスケーラブルな環境で、パブリックサブネットが 1 つある場合は、追加の設定は必要ありません。これは、この環境では、Amazon EC2 インスタンスがパブリック IP アドレスで設定され、これによりインスタンスがインターネットと通信できるようになるからです。

Amazon VPC で Elastic Beanstalk を使用する方法については、「Amazon VPC で Elastic Beanstalk を使用する」を参照してください。

Elastic Beanstalk アプリケーションの管理

この章では、Elastic Beanstalk アプリケーションを管理および設定する方法について説明します。を使用する最初のステップ AWS Elastic Beanstalk は、ウェブアプリケーションを表すアプリケーションを作成することです AWS。Elastic Beanstalk では、アプリケーションは、ウェブアプリケーションを実行する環境、Elastic Beanstalk の使用時に作成するウェブアプリケーションのソースコードのバージョン、保存されている設定、ログ、その他のアーティファクトなどのコンテナとして機能します。

アプリケーションを作成する方法

1. Elastic Beanstalk 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、[Applications] (アプリケーション) を選択し、[Create application] (アプリケーションの作成) を選択します。
3. 画面上のフォームを使用して、アプリケーション名を指定します。
4. (オプション) 説明を入力し、タグのキーと値を追加します。
5. [作成] を選択します。

アプリケーションの作成後に、このアプリケーションの環境を作成することをコンソールから求められます。利用できるすべてのオプションについての詳細は、「Elastic Beanstalk 環境の作成」を参照してください。

アプリケーションが不要になった場合は、削除できます。

> **Warning**
> アプリケーションを削除すると、すべての関連付けられた環境が終了され、そのアプリケーションに属するすべてのアプリケーションバージョンと保存済み設定が削除されます。

アプリケーションを削除するには

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションを選択します。
3. [アクション] を選択してから、[アプリケーションの削除] を選択します。

トピック

- Elastic Beanstalk アプリケーション管理コンソール
- アプリケーションバージョンの管理
- Elastic Beanstalk アプリケーションソースバンドルを作成する
- での EB CLI の使用 AWS CodeBuild
- アプリケーションのタグ付け
- Elastic Beanstalk アプリケーションリソースのタグ付け

Elastic Beanstalk アプリケーション管理コンソール

このトピックでは、 AWS Elastic Beanstalk コンソールを使用してアプリケーション、アプリケーションバージョン、および保存された設定を管理する方法について説明します。

アプリケーション管理コンソールにアクセスするには

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。

   アプリケーションの概要ページには、アプリケーションに関連付けられているすべての環境の概要が一覧表示されます。
3. 続行するにはいくつかの方法があります。

   a. [アクション] ドロップダウンメニューから、次のいずれかのアプリケーション管理アクションを選択できます。[環境の作成]、[アプリケーションの削除]、[アプリケーションバージョンの表示]、[保存済み設定の表示]、[終了した環境の復元]。

      このアプリケーションで環境を起動するには、[環境の作成] を直接選択します。詳細については、「the section called “環境の作成”」を参照してください。

   b. このページでは、環境にデプロイされているアプリケーションの横に環境名が表示されます。名前で環境を選択して、その環境の環境管理コンソールに移動します。ここで、環境を設定、モニタリング、または管理することができます。

   c. リストからアプリケーションを選択すると、左側のナビゲーションペインにアプリケーションが一覧表示されます。

- アプリケーションのアプリケーションバージョンを表示および管理するには、ナビゲーションペインでアプリケーション名の後にある [アプリケーションバージョン] を選択します。

  アプリケーションバージョンは、アプリケーションコードのアップロードされたバージョンです。新しいバージョンのアップロード、既存のバージョンの任意のアプリケーション環境へのデプロイ、または古いバージョンの削除を行うことができます。詳細については、「アプリケーションバージョンの管理」を参照してください。
- 実行中の環境から保存された設定を表示および管理するには、ナビゲーションペインのアプリケーション名の後にある [保存された設定] を選択します。

  保存された設定は、環境の設定を以前の状態に戻したり、同じ設定で新しい環境を作成したりするために使用できる設定のコレクションです。詳細については、「Elastic Beanstalk の保存された設定を使用する」を参照してください。

アプリケーションバージョンの管理

このトピックでは、アプリケーションバージョンとその作成および管理方法について説明します。

Elastic Beanstalk は、ソースコードをアップロードするたびにアプリケーションバージョンを作成します。これは通常、環境管理コンソールまたは EB CLI を使用して環境を作成するか、コードをアップロードしてデプロイするときに発生します。Elastic Beanstalk は、アプリケーションのライフサイクルポリシーおよびアプリケーションの削除時に、これらのアプリケーションバージョンを削除します。アプリケーションライフサイクルポリシーの詳細については、「アプリケーションバージョンライフサイクルの設定」を参照してください。

また、アプリケーション管理コンソールまたは EB CLI コマンド eb appversion を使用して、デプロイすることなくソースバンドルをアップロードすることもできます。Elastic Beanstalk によって、ソースバンドルは Amazon Simple Storage Service (Amazon S3) に保存されます。自動的に削除されることはありません。

新しいアプリケーションバージョンの作成時にタグを適用できます。また、既存のアプリケーションバージョンのタグを編集できます。詳細については、「アプリケーションバージョンのタグ付け」を参照してください。

アプリケーションバージョンの作成

EB CLI を使用して新しいアプリケーションバージョンを作成することもできます。詳細については、EB CLI コマンドのチャプターの「eb appversion」を参照してください。

> **Note**
> 時間の経過とともに、アプリケーションで、多数のアプリケーションバージョンが累積されることがあります。ストレージ領域を節約し、アプリケーションバージョンのクォータに達しないようにするには、不要になったアプリケーションバージョンを削除することをお勧めします。

次の手順で指定したファイルは、アプリケーションに関連付けられます。アプリケーションのバージョンを、新しい環境または既存の環境にデプロイできます。

新しいアプリケーションバージョンを作成するには

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. ナビゲーションペインで、アプリケーション名を見つけ、[アプリケーションバージョン] を選択します。
4. [アップロード] を選択します。画面上のフォームを使用して、アプリケーションのソースバンドルをアップロードします。

   > **Note**
   > ソースバンドルのファイルサイズの上限は 500 MB です。

5. 必要に応じて、簡単な説明を入力し、タグのキーと値を追加します。
6. [アップロード] を選択します。

アプリケーションバージョンの削除

EB CLI を使用してアプリケーションバージョンを削除することもできます。詳細については、EB CLI コマンドのチャプターの「eb appversion」を参照してください。

> **Note**
> アプリケーションバージョンを削除しても、現在そのバージョンで実行されている環境に影響はありません。

また、アプリケーションバージョンライフサイクルの設定を定義して、古いバージョンを自動的に削除するように Elastic Beanstalk を設定することもできます。これらのライフサイクルを設定すると、新しいアプリケーションバージョンを作成するときにそれが適用されます。例えば、アプリケーションバージョンの最大数を 25 に設定している場合、26 番目のバージョンがアップロードされると Elastic Beanstalk によって最も古いバージョンが削除されます。90 日間の最長有効期間を設定すると、新しいバージョンがアップロードされた際に 90 日を超えているものが削除されます。詳細については、「the section called “バージョンライフサイクル”」を参照してください。

アプリケーションバージョンを削除するには

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. ナビゲーションペインで、アプリケーション名を見つけ、[アプリケーションバージョン] を選択します。
4. 削除するアプリケーションバージョンを 1 つ以上選択します。
5. [アクション] を選択してから、[削除] をクリックします。
6. (オプション) これらのアプリケーションバージョンのアプリケーションソースバンドルを Amazon Simple Storage Service (Amazon S3) バケットに残すには、[Delete versions from Amazon S3 (Amazon S3 からのバージョンの削除)] ボックスをオフにします。
7. [削除] を選択します。

Amazon S3 からソースバンドルを削除することを選択しなかった場合でも、Elastic Beanstalk によってそのバージョンはレコードから削除されます。ただし、ソースバンドルは Elastic Beanstalk ストレージバケットに残ります。アプリケーションバージョンのクォータは Elastic Beanstalk の追跡対象のバージョンにのみ適用されます。したがって、必要ならば、クォータ内に収まるようにバージョンを削除できますが、すべてのソースバンドルは Amazon S3 で保持することができます。

> **Note**
>
> アプリケーションバージョンのクォータはソースバンドルには適用されませんが、依然として Amazon S3 料金が発生し、必要な時間を超えて個人情報が保持されることがあります。Elastic Beanstalk によって、ソースバンドルが自動的に削除されることはありません。必要がなくなったらソースバンドルを削除する必要があります。

アプリケーションバージョンライフサイクルの設定

このトピックでは、Elastic Beanstalk が特定の環境内のアプリケーションのバージョンに適用するポリシーとクォータについて説明します。これには、アプリケーションバージョンが環境に存在する期間の長さも含まれます。

Elastic Beanstalk コンソールまたは EB CLI を使用してアプリケーションの新しいバージョンをアップロードするたびに、Elastic Beanstalk によってアプリケーションバージョンが作成されます。使用しなくなったバージョンを削除しないと、最終的にはアプリケーションバージョンのクォータに到達し、そのアプリケーションの新しいバージョンを作成できなくなります。

アプリケーションにアプリケーションバージョンライフサイクルポリシーを適用することで、クォータに到達するのを回避できます。ライフサイクルポリシーにより、古いアプリケーションバージョンを削除するか、アプリケーションの合計数が指定した数を超えた場合にアプリケーションバージョンを削除するよう Elastic Beanstalk に指示されます。

Elastic Beanstalk は、新しいアプリケーションバージョンを作成するたびにアプリケーションのライフサイクルポリシーを適用し、ライフサイクルポリシーが適用されるたびに、最大 100 個のバージョンを削除します。Elastic Beanstalk は、新しいバージョンを作成した後で旧バージョンを削除します。新しいバージョンは、ポリシーで定義した最大バージョン数にはカウントされません。

Elastic Beanstalk は、環境で現在使用されているアプリケーションバージョン、またはポリシーがトリガーされる 10 週前までに終了した環境にデプロイされているアプリケーションバージョンは削除しません。

アプリケーションバージョンのクォータはリージョン内のすべてのアプリケーションに適用されます。複数のアプリケーションがある場合は、クォータに達するのを回避するため、各アプリケーションを適切なライフサイクルポリシーで設定します。例えば、リージョンに 10 個のアプリケーションがあり、クォータが 1,000 アプリケーションバージョンである場合、すべてのアプリケーションについて、クォータを 99 アプリケーションにしたライフサイクルポリシーを設定するか、合計が 1,000 アプリケーションバージョン未満である限り、各アプリケーションにその他の値を設定するこ

とを検討してください。Elastic Beanstalk は、アプリケーションバージョンの作成に成功した場合のみポリシーを適用するため、既にクォータに達している場合は、新しいバージョンを作成する前に、いくつかのバージョンを手動で削除する必要があります。

デフォルトでは、Elastic Beanstalk は、データの損失を防ぐために、アプリケーションバージョンのソースバンドルを Amazon S3 に残します。ソースバンドルを削除すると、領域を節約できます。

Elastic Beanstalk CLI および API を介してライフサイクル設定を定義できます。詳細については、「eb appversion」、「CreateApplication」(`ResourceLifecycleConfig` パラメータを使用)、「UpdateApplicationResourceLifecycle」を参照してください。

コンソールでアプリケーションのライフサイクル設定を指定する

Elastic Beanstalk コンソールでライフサイクル設定を指定できます。

アプリケーションのライフサイクル設定を指定するには

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. ナビゲーションペインで、アプリケーション名を見つけ、[アプリケーションバージョン] を選択します。
4. [設定] を選択します。
5. 画面上のフォームを使用して、アプリケーションのライフサイクル設定を構成します。
6. [保存] を選択します。

[設定] ページでは以下の操作を実行できます。

- アプリケーションバージョンの合計数またはアプリケーションバージョンの期間に基づいて、ライフサイクル設定を構成します。
- アプリケーションバージョンが削除されるときにソースバンドルを S3 から削除するかどうかを指定します。
- アプリケーションバージョンの削除に使用するロールを指定します。バージョン削除に必要なすべてのアクセス許可を含めるには、`aws-elasticbeanstalk-service-role` という名前のデフォルトの Elastic Beanstalk サービスロールを選択するか、Elastic Beanstalk 管理サービスポリシーを使用して別のサービスロールを選択します。詳細については、「Elastic Beanstalk サービスロールの管理」を参照してください。

アプリケーションバージョンのタグ付け

このトピックでは、Elastic Beanstalk アプリケーションバージョンにタグ付けする利点と、タグを管理する方法について説明します。

AWS Elastic Beanstalk アプリケーションバージョンにタグを適用できます。タグは、 AWS リソースに関連付けられたキーと値のペアです。Elastic Beanstalk リソースのタグ付け、ユースケース、タグのキーと値の制約、サポートされているリソースタイプの詳細については、「Elastic Beanstalk アプリケーションリソースのタグ付け」を参照してください。

アプリケーションバージョンを作成するときにタグを指定できます。既存のアプリケーションバージョンでは、タグの追加や削除、既存タグの値の更新ができます。アプリケーションバージョンごとに最大 50 個のタグを追加できます。

アプリケーションバージョンの作成時にタグを追加する

Elastic Beanstalk コンソールを使用して環境を作成する際に、アプリケーションコードのバージョンをアップロードすることを選択すると、新しいアプリケーションバージョンに関連付けるタグのキーと値を指定できます。

Elastic Beanstalk コンソールを使用して、環境ですぐには使用しないアプリケーションバージョンをアップロードすることもできます。アプリケーションバージョンをアップロードするときに、タグキーと値を指定できます。

AWS CLI または他の API ベースのクライアントで、 create-application-version コマンドの `--tags`パラメータを使用してタグを追加します。

```
$ aws elasticbeanstalk create-application-version \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --application-name my-app --version-label v1
```

EB CLI を使用して環境を作成または更新すると、デプロイしたコードからアプリケーションバージョンが作成されます。EB CLI での作成時にアプリケーションバージョンに直接タグ付けする方法はありません。既存のアプリケーションバージョンにタグを追加する方法については、次のセクションを参照してください。

既存のアプリケーションバージョンのタグを管理する

既存の Elastic Beanstalk アプリケーションバージョンでタグを追加、更新、削除できます。

Elastic Beanstalk コンソールを使用してアプリケーションバージョンのタグを管理するには

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. ナビゲーションペインで、アプリケーション名を見つけ、[アプリケーションバージョン] を選択します。
4. 管理するアプリケーションバージョンを選択します。
5. [Actions (アクション)] を選択して [Manage tags (タグの管理)] を選択します。
6. 画面上のフォームを使用して、タグを追加、更新、または削除します。
7. ページの最下部で [適用] を選択し変更を保存します。

EB CLI を使用してアプリケーションバージョンを更新する場合は、 eb tags を使用してタグを追加、更新、削除、一覧表示します。

たとえば、次のコマンドでは、アプリケーションバージョンのタグを一覧表示します。

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:applicationversion/my-app/my-version"
```

次のコマンドでは、`mytag1` タグを更新して `mytag2` タグを削除します。

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:applicationversion/my-app/my-version"
```

オプションの完全なリストおよび詳細な例については、「`eb tags`」を参照してください。

AWS CLI または他の API ベースのクライアントでは、 list-tags-for-resource コマンドを使用してアプリケーションバージョンのタグを一覧表示します。

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:applicationversion/my-app/my-version"
```

アプリケーションバージョンのタグを追加、更新、または削除するには、update-tags-for-resource コマンドを使用します。

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-
id:applicationversion/my-app/my-version"
```

追加するタグと更新するタグを update-tags-for-resource の `--tags-to-add` パラメータで指定します。存在していないタグが追加され、既存のタグの値が更新されます。

> **Note**
>
> 一部の EB CLI と AWS CLI コマンドを Elastic Beanstalk アプリケーションバージョンで使用するには、アプリケーションバージョンの ARN が必要です。ARN を取得するには、次のコマンドを使用します。
>
> ```
> $ aws elasticbeanstalk describe-application-versions --application-name my-app
> --version-label my-version
> ```

Elastic Beanstalk アプリケーションソースバンドルを作成する

このトピックでは、ソースバンドル内の Elastic Beanstalk にアプリケーションソースファイルをアップロードする方法について説明します。ソースバンドルの要件、構造、および作成方法について説明します。

AWS Elastic Beanstalk コンソールを使用して新しいアプリケーションまたはアプリケーションバージョンをデプロイする場合は、アプリケーションのファイルをソースバンドルにアップロードする必要があります。ソースバンドルは次の要件を満たしている必要があります。

- 単一の `ZIP` ファイルまたは `WAR` ファイルで構成される (`WAR` ファイル内に複数の `ZIP` ファイルを含めることが可能)
- 500 MB 以下
- 親フォルダまたは最上位ディレクトリを含まない (サブディレクトリを除く)

定期的なバックグラウンドタスクを処理するワーカーアプリケーションをデプロイする場合には、アプリケーションソースバンドルに `cron.yaml` ファイルも含める必要があります。詳細については、「定期的なタスク」を参照してください。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI)、 AWS Toolkit for Eclipse、または AWS Toolkit for Visual Studio を使用してアプリケーションをデプロイする場合、ZIP ファイルまたは WAR ファイルは自動的に正しく構造化されます。詳細については Elastic Beanstalk を管理するための EB コマンドラインインターフェイス (EB CLI) の設定、Elastic Beanstalk への Java アプリケーションのデプロイ、および AWS Toolkit for Visual Studio を参照してください。

セクション

- コマンドラインからソースバンドルを作成する
- Git を使用してソースバンドルを作成する
- Mac OS X Finder または Windows エクスプローラでファイルを圧縮する
- .NET アプリケーションのソースバンドルの作成
- ソースバンドルをテストする

コマンドラインからソースバンドルを作成する

`zip` コマンドを使用してソースバンドルを作成します。非表示のファイルとフォルダを含めるには、次のようなパターンを使用します。

```
~/myapp$ zip ../myapp.zip -r * .[^.]*
  adding: app.js (deflated 63%)
  adding: index.js (deflated 44%)
  adding: manual.js (deflated 64%)
  adding: package.json (deflated 40%)
  adding: restify.js (deflated 85%)
  adding: .ebextensions/ (stored 0%)
  adding: .ebextensions/xray.config (stored 0%)
```

これにより、Elastic Beanstalk 設定ファイルと、ピリオドで始まるその他のファイルとフォルダがアーカイブに含まれます。

Tomcat ウェブアプリケーションの場合は、`jar` を使用してウェブアーカイブを作成します。

```
~/myapp$ jar -cvf myapp.war .
```

上記のコマンドには、ソースバンドルのサイズを不要に増やす可能性のある非表示のファイルが含まれます。よりきめ細かに制御するには、より詳細なファイルパターンを使用するか、Git でソースバンドルを作成します。

Git を使用してソースバンドルを作成する

Git を使用してアプリケーションのソースコードを管理している場合、`git archive` コマンドを使用してソースバンドルを作成します。

```
$ git archive -v -o myapp.zip --format=zip HEAD
```

`git archive` には、Git に保存されているファイルのみが含まれ、無視されたファイルや Git ファイルは除外されます。これにより、ソースバンドルをできるだけ小さくできます。詳細については、git-archive のマニュアルページを参照してください。

Mac OS X Finder または Windows エクスプローラでファイルを圧縮する

Mac OS X Finder または Windows エクスプローラーで ZIP ファイルを作成する場合、親フォルダではなく、ファイルとサブフォルダを圧縮していることを確認します。

> **Note**
>
> Mac OS X および Linux ベースのオペレーティング システムのグラフィカルユーザーインターフェイス (GUI) は、ピリオド (.) で始まる名前のファイルとフォルダを表示しません。ZIP ファイルに `.ebextensions` などの隠しフォルダを含める必要がある場合、アプリケーションの圧縮に、GUI ではなくコマンドラインインターフェイスを使用します。Mac OS X または Linux ベースのオペレーティング システムで ZIP ファイルを作成するコマンドラインの手順については、「コマンドラインからソースバンドルを作成する」を参照してください。

Example

`myapp` という Python プロジェクトフォルダがあり、そこに次のファイルとサブフォルダが含まれるとします。

```
myapplication.py
README.md
static/
static/css
static/css/styles.css
static/img
```

```
static/img/favicon.ico
static/img/logo.png
templates/
templates/base.html
templates/index.html
```

上記の要件一覧に示すように、ソースバンドルは親フォルダなしで圧縮し、解凍後の構造には余分な上位レベルのディレクトリが含まれないようにする必要があります。この例では、ファイルの圧縮時に `myapp` フォルダは作成されません（または、コマンドラインで `myapp` セグメントがファイルパスに追加されません）。

このサンプルファイル構造はこのトピックを通して使用され、ファイルを圧縮する方法を示します。

.NET アプリケーションのソースバンドルの作成

Visual Studio を使用する場合は、 に含まれているデプロイツールを使用して AWS Toolkit for Visual Studio 、.NET アプリケーションを Elastic Beanstalk にデプロイできます。詳細については、「デプロイメントツールを使用した Elastic Beanstalk アプリケーションの.NET でのデプロイ」を参照してください。

.NET アプリケーションのソースバンドルを手動で作成する必要がある場合は、プロジェクトディレクトリを含む `ZIP` ファイルを簡単に作成することはできません。Elastic Beanstalk へのデプロイに適したプロジェクトのウェブデプロイパッケージを作成する必要があります。デプロイパッケージを作成するために使用できる方法にはいくつかあります。

- Visual Studio の Web 発行 ウィザードを使用してデプロイパッケージを作成します。詳細については、Visual Studio でウェブデプロイパッケージを作成する方法を参照してください。

  > **Important**
  >
  > ウェブデプロイパッケージを作成するとき、サイト名 は `Default Web Site` で始める必要があります。

- .NET プロジェクトを用意している場合、次の例に示すように、msbuild コマンドを使用してデプロイパッケージを作成できます。

  > **Important**
  >
  > `DeployIisAppPath` パラメータは `Default Web Site` で始める必要があります。

```
C:/> msbuild <web_app>.csproj /t:Package /p:DeployIisAppPath="Default Web Site"
```

- ウェブサイトプロジェクトを用意している場合、IIS ウェブデプロイツールを使用してデプロイパッケージを作成します。詳細については、Packaging and Restoring a Web site を参照してください。

> **Important**
>
> `apphostconfig` パラメータは `Default Web Site` で始める必要があります。

複数のアプリケーションまたは ASP.NET コアアプリケーションをデプロイする場合は、ソースバンドルのルートに、`.ebextensions` フォルダをアプリケーションバンドルとマニフェストファイルに並べて置きます。

```
~/workspace/source-bundle/
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- AspNetCore101HelloWorld.zip
|-- AspNetCoreHelloWorld.zip
|-- aws-windows-deployment-manifest.json
`-- VS2015AspNetWebApiApp.zip
```

ソースバンドルをテストする

ソースバンドルを Elastic Beanstalk にアップロードする前に、ローカルでテストする必要があります。Elastic Beanstalk は基本的にファイルの抽出にコマンドラインを使用するため、GUI ツールではなくコマンドラインからテストを実行することをお勧めします。

解凍されたファイルが、新しい上位フォルダまたはディレクトリではなく、アーカイブ自体があるフォルダに表示されていることを確認します。

での EB CLI の使用 AWS CodeBuild

AWS CodeBuild はソースコードをコンパイルし、単体テストを実行して、すぐにデプロイできるアーティファクトを生成します。CodeBuild を EB CLI をともに使用すれば、ソースコードからのア

プリケーションの構築を自動化できます。環境の作成とその後のデプロイはビルドステップで始まり、続いて生成されるアプリケーションがデプロイされます。

> **Note**
>
> 一部のリージョンでは、CodeBuild を使用できない場合があります。これらのリージョンでは、Elastic Beanstalk と CodeBuild を統合できません。
> 各リージョンで提供される AWS サービスの詳細については、「リージョンテーブル」を参照してください。

Creating an application

CodeBuild を使用する Elastic Beanstalk アプリケーションを作成するには

1. アプリケーションフォルダに、CodeBuild ビルド仕様ファイル `buildspec.yml` を含めます。
2. Elastic Beanstalk に固有のオプションを持つ `eb_codebuild_settings` エントリをファイルに追加します。
3. フォルダで eb init を実行します。

> **Note**
>
> EB CLI を CodeBuild と共に使用する際には、アプリケーション名にピリオド (`.`) やスペース ( ) を使用しないでください。

Elastic Beanstalk は、CodeBuild ビルド仕様ファイル形式を拡張して、次の追加設定を含めます。

```
eb_codebuild_settings:
  CodeBuildServiceRole: role-name
  ComputeType: size
  Image: image
  Timeout: minutes
```

`CodeBuildServiceRole`

CodeBuild がユーザーに代わって依存 AWS サービスとやり取りするために使用できる AWS Identity and Access Management (IAM) サービスロールの ARN または名前。この値は必須です。これを省略すると、それ以降の eb create または eb deploy コマンドは失敗します。

CodeBuild のサービスロールの作成に関する詳細については、「AWS CodeBuild ユーザーガイド」の「CodeBuild サービスロールの作成」を参照してください。

> **Note**
>
> CodeBuild 自体でアクションを実行するためのアクセス許可も必要です。Elastic Beanstalk の管理ユーザーポリシー、AdministratorAccess-AWSElasticBeanstalk には、CodeBuild アクションが必要とする、すべてのアクセス許可が含まれています。管理ポリシーを使用していない場合は、ユーザーポリシーで必ず以下のアクセス権限を許可してください。
>
> ```
> "codebuild:CreateProject",
> "codebuild:DeleteProject",
> "codebuild:BatchGetBuilds",
> "codebuild:StartBuild"
> ```
>
> 詳細については、「Elastic Beanstalk ユーザーポリシーの管理」を参照してください。

`ComputeType`

CodeBuild ビルド環境で Docker コンテナによって使用されるリソースの量。有効な値は、BUILD_GENERAL1_SMALL、BUILD_GENERAL1_MEDIUM、BUILD_GENERAL1_LARGE です。

`Image`

CodeBuild がビルド環境に使用する Docker Hub または Amazon ECR イメージの名前。この Docker イメージには、コードを作成するために必要なすべてのツールとランタイムライブラリが含まれている必要があります。また、イメージは、アプリケーションのターゲットプラットフォームに一致する必要があります。CodeBuild では、特に Elastic Beanstalk での使用を意図した一連のイメージが管理および保守されています。それらのイメージのいずれかを使用することをお勧めします。詳細については、「AWS CodeBuild ユーザーガイド」の「CodeBuild に用意されている Docker イメージ」を参照してください。

`Image` 値はオプションです。これを省略すると、eb init コマンドはターゲットプラットフォームに最適なイメージの選択を試みます。さらに、インタラクティブモードで eb init で実行し、イメージの選択に失敗した場合、イメージの選択が求められます。初期化が正常に終了すると、eb init は選択されたイメージを `buildspec.yml` ファイルに書き込みます。

`Timeout`

CodeBuild ビルドがタイムアウトするまでの実行時間 (分)。この値はオプションです。有効な値とデフォルト値の詳細については、「CodeBuild でのビルドプロジェクトの作成」を参照してください。

> **Note**
>
> このタイムアウトにより、CodeBuild の最大実行時間が制御されます。また、EB CLI はアプリケーションバージョンを作成するための最初のステップの一部としてこれを順守します。これは、eb create または eb deploy コマンドの `--timeout` オプションで指定する値とは異なります。後者の値は、EB CLI が環境の作成または更新を待機する最大時間を制御します。

アプリケーションコードのビルドとデプロイ

アプリケーションコードをデプロイする必要が生じると、EB CLI は常に CodeBuild を使用してビルドを実行し、生成されたビルドアーティファクトを環境にデプロイします。この処理は、eb create コマンドを使用してアプリケーションの Elastic Beanstalk 環境を作成するときと、後で eb deploy コマンドを使用してコードの変更を環境にデプロイするたびに発生します。

CodeBuild のステップが失敗した場合、環境の作成またはデプロイは開始されません。

アプリケーションのタグ付け

このトピックでは、Elastic Beanstalk アプリケーションのタグ付けの利点について説明します。また、アプリケーションタグを作成および管理する手順も示します。タグは、 AWS リソースに関連付けられたキーと値のペアです。Elastic Beanstalk リソースのタグ付け、ユースケース、タグのキーと値の制約、サポートされているリソースタイプの詳細については、「Elastic Beanstalk アプリケーションリソースのタグ付け」を参照してください。

アプリケーションを作成するときにタグを指定できます。既存のアプリケーションでは、タグの追加や削除、既存タグの値の更新ができます。各アプリケーションには、最大 50 個のタグを追加できます。

アプリケーションの作成時にタグを追加する

Elastic Beanstalk コンソールを使用してアプリケーションを作成するときに、[Create New Application (新しいアプリケーションの作成)] ダイアログボックスでタグのキーと値を指定できます。

EB CLI を使用して環境を作成する場合は。eb init の `--tags` オプションを使用してタグを追加します。

```
~/workspace/my-app$ eb init --tags mytag1=value1,mytag2=value2
```

AWS CLI または他の API ベースのクライアントで、 create-application コマンドの `--tags`パラメータを使用してタグを追加します。

```
$ aws elasticbeanstalk create-application \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --application-name my-app --version-label v1
```

既存のアプリケーションのタグを管理する

既存の Elastic Beanstalk アプリケーションでタグを追加、更新、削除できます。

Elastic Beanstalk コンソールでアプリケーションのタグを管理するには

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. [Actions (アクション)] を選択して [Manage tags (タグの管理)] を選択します。
4. 画面上のフォームを使用して、タグを追加、更新、または削除します。
5. ページの最下部で [適用] を選択し変更を保存します。

EB CLI を使用してアプリケーションを更新する場合は、eb tags を使用してタグを追加、更新、削除、一覧表示します。

たとえば、次のコマンドでは、アプリケーションのタグを一覧表示します。

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:application/my-app"
```

次のコマンドでは、mytag1 タグを更新して mytag2 タグを削除します。

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:application/my-app"
```

オプションの完全なリストおよび詳細な例については、「eb tags」を参照してください。

AWS CLI または他の API ベースのクライアントでは、 list-tags-for-resource コマンドを使用してアプリケーションのタグを一覧表示します。

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:application/my-app"
```

アプリケーションのタグを追加、更新、または削除するには、update-tags-for-resource コマンドを使用します。

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:application/my-
app"
```

追加するタグと更新するタグを update-tags-for-resource の `--tags-to-add` パラメータで指定します。存在していないタグが追加され、既存のタグの値が更新されます。

> **Note**
>
> Elastic Beanstalk アプリケーションで一部の EB CLI と AWS CLI コマンドを使用するには、アプリケーションの ARN が必要です。ARN を取得するには、次のコマンドを使用します。
>
> ```
> $ aws elasticbeanstalk describe-applications --application-names my-app
> ```

Elastic Beanstalk アプリケーションリソースのタグ付け

このトピックでは、Elastic Beanstalk アプリケーションリソースでタグを使用する利点と、その制約について説明します。また、アプリケーションリソースのタグを作成および管理する方法についても説明します。

AWS Elastic Beanstalk アプリケーションのリソースにタグを適用できます。タグは、 AWS リソースに関連付けられたキーと値のペアです。タグは、リソースを分類するのに役立ちます。これらは、複数の AWS アプリケーションの一部として多くのリソースを管理する場合に特に便利です。

Elastic Beanstalk のリソースにタグ付けする場合に使用するいくつかの方法を以下に示します。

- デプロイのステージ – アプリケーションのさまざまなステージ (開発、ベータ、本稼働など) に関連付けられているリソースを特定します。
- コスト配分 – コスト配分レポートを使用して、さまざまな経費アカウントに関連付けられた AWS リソースの使用状況を追跡します。レポートにはタグ付きのリソースとタグが付いていないリソースの両方が含まれ、タグに従ってコストが集計されます。コスト割り当てレポートでタグがどのように使用されているかについては、AWS 請求とコスト管理ユーザーガイドのコスト割り当てタグを使用したカスタム請求レポートを参照してください。
- アクセスコントロール – タグを使用して、リクエストとリソースに対するアクセス許可を管理します。たとえば、ベータ環境の作成と管理のみができるユーザーには、ベータステージのリソースにのみアクセスを許可します。詳細については、「タグを使用した Elastic Beanstalk リソースへのアクセスのコントロール」を参照してください。

リソースごとに最大 50 個のタグを追加できます。各環境はわずかに異なります。Elastic Beanstalk は各環境に 3 つのデフォルトのシステムタグを追加します。これらのタグは編集または削除できません。デフォルトのタグに加え、最大 47 個のタグを各環境に追加できます。

タグのキーと値には以下の制約が適用されます。

- キーおよび値には、文字、数字、空白、および記号 `_ . : / = + - @` を使用できます。
- キーの長さは最大 127 文字です。値の長さは最大 255 文字です。

> **Note**
>
> これらの長さの制限は、UTF-8 の Unicode 文字に対するものです。他のマルチバイトエンコードに対しては、制限がより低くなる場合があります。

- キーでは、大文字と小文字が区別されます。
- キーは `aws:` または `elasticbeanstalk:` で始めることはできません。

タグを付けることができるリソース

タグ付けできる Elastic Beanstalk リソースのタイプは以下のとおりです。タイプ別のタグの管理に関するトピックへのリンクも示します。

- アプリケーション
- 環境
- アプリケーションバージョン
- 保存された設定

起動テンプレートへのタグの伝播

Elastic Beanstalk には、起動テンプレートへの環境タグの伝播を有効にするオプションがあります。このオプションでは、起動テンプレートによるタグベースのアクセスコントロール (TBAC) のサポートを継続しています。

> **Note**
>
> 起動設定は段階的に廃止され、起動テンプレートに置き換えられています。詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「起動構成」を参照してください。

実行中の EC2 インスタンスのダウンタイムを防ぐため、タグ AWS CloudFormation は既存の起動テンプレートに伝達されません。環境のリソースにタグが必要なユースケースがある場合は、Elastic Beanstalk を有効にして、これらのリソースのタグを含む起動テンプレートを作成できます。これを行うには、aws:autoscaling:launchconfiguration 名前空間の `LaunchTemplateTagPropagationEnabled` オプションを `true` に設定します。デフォルト値は `false` です。

次の設定ファイルの例では、起動テンプレートへのタグの伝播を有効にしています。

```
option_settings:
  aws:autoscaling:launchconfiguration:
    LaunchTemplateTagPropagationEnabled: true
```

Elastic Beanstalk は、次のリソースの起動テンプレートにのみタグを伝播できます。

- EBS ボリューム

- EC2インスタンス
- EC2 ネットワークインターフェイス
- AWS CloudFormation リソースを定義する起動テンプレート

CloudFormation では特定のリソースのテンプレート作成時にのみタグが許可されるため、この制約が存在します。詳細については、「AWS CloudFormation ユーザーガイド」の「TagSpecification」を参照してください。

> **Important**
>
> - 既存の環境でこのオプション値を `false` から `true` に変更すると、既存のタグに互換性を破る変更が生じる可能性があります。
> - この機能を有効にすると、タグの伝播に EC2 の置き換えが必要になり、ダウンタイムが発生する可能性があります。ローリング更新を有効にして設定の変更を一括で適用し、更新プロセス中のダウンタイムを防ぐことができます。詳細については、「設定変更」を参照してください。

起動テンプレートの詳細については、以下を参照してください。

- 「Amazon EC2 Auto Scaling ユーザーガイド」の「起動テンプレート」
- 「AWS CloudFormation ユーザーガイド」の「テンプレートの操作」
- 「AWS CloudFormation ユーザーガイド」の「Elastic Beanstalk テンプレートスニペット」

Elastic Beanstalk での環境の作成

この章では、Elastic Beanstalk 環境を作成して管理する方法について説明します。この入門ページでは、アプリケーションと環境の進化に伴って適用される更新、メンテナンス、設定の概要を説明します。

環境機能

開発環境やテスト環境、本稼働環境を個別に作成して管理し、アプリケーションの任意のバージョンを環境にデプロイすることができます。環境は長期的なものでも一時的なものでも構いません。環境を終了する際に設定を保存しておけば、後で再作成することも可能です。

アプリケーションのデプロイ

アプリケーションの開発では、目的別に異なる環境にデプロイすることが多々あります。Elastic Beanstalk では、デプロイの実行方法を設定できます。このような場合、環境内にあるすべてのインスタンスにアプリケーションを同時にデプロイしたり、ローリングデプロイでデプロイをバッチに分割することができます。

設定変更

設定変更は、デプロイとは別に処理され、独自の適用範囲があります。たとえば、アプリケーションを実行している EC2 インスタンスのタイプを変更する場合は、すべてのインスタンスを置き換えなければなりません。一方、環境のロードバランサーの設定を変更する場合、サービスを中断したり容量を減らすことなくインプレースで行えます。環境内のインスタンスに影響を与える設定変更は、ローリング設定更新によりバッチ単位で適用することも可能です。

> **Note**
>
> 環境内にあるリソースを変更する場合は、必ず Elastic Beanstalk を使用してください。他のサービスのコンソール、CLI コマンド、または SDK を使用してリソースを変更した場合、Elastic Beanstalk ではこれらのリソースの状態を正確にモニタリングできなくなります。また、設定を保存することも、環境を確実に再現することもできなくなります。帯域外で変更を加えた場合は、環境を更新または終了する際も問題が発生することがあります。

プラットフォームの更新

環境を起動する際には、まずプラットフォームのバージョンを選択します。パフォーマンスを向上したり新機能を追加するため、プラットフォームは新しいプラットフォームのバージョンで定期的に更

新されます。いつでもお客様の環境を最新のプラットフォームのバージョンに更新できます。サポートされているプラットフォームのリストと、最新版だった日付範囲を含むプラットフォームバージョンの履歴については、「AWS Elastic Beanstalk プラットフォーム」ガイドを参照してください。

アーキテクチャオプション

アプリケーションが複雑になるにつれて、複数のコンポーネントに分割して、各コンポーネントが個別の環境で実行されるようにできます。長時間実行のワークロード用には、Amazon Simple Queue Service (Amazon SQS) キューからのジョブを処理するワーカー環境を起動できます。

トピック

- Elastic Beanstalk 環境マネジメントコンソールを使用する
- Elastic Beanstalk 環境の作成
- EB CLI で複数の Elastic Beanstalk 環境をグループとして管理する
- Elastic Beanstalk 環境へのアプリケーションのデプロイ
- 設定変更
- Elastic Beanstalk 環境のプラットフォームバージョンの更新
- 環境設定の更新およびアプリケーションのデプロイのキャンセル
- Elastic Beanstalk 環境の再構築
- 環境タイプ
- Elastic Beanstalk ワーカー環境
- Elastic Beanstalk 環境間のリンクの作成
- Elastic Beanstalk 環境を無効な状態から復旧する

Elastic Beanstalk 環境マネジメントコンソールを使用する

このセクションでは、環境管理コンソールを使用して Elastic Beanstalk 環境を管理する方法について説明します。コンソールには、環境の設定を管理し、一般的なアクションを実行する機能があります。これらのアクションには、環境で実行されているウェブサーバーの再起動、環境のクローン作成、または環境を最初から再構築することが含まれます。

トピック

- 環境マネジメントコンソールへのアクセス
- 環境の概要のペイン

- 環境の詳細
- 環境アクション

環境マネジメントコンソールへのアクセス

次に示すのは、環境管理コンソールを起動する手順です。

Elastic Beanstalk コンソールに既にログインしている場合は、アプリケーション管理コンソール から環境管理ページを起動することもできます。リストから環境を選択すると、選択した環境の管理コンソールの詳細が表示されます。

環境マネジメントコンソールにアクセスするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。

次の図は、環境管理コンソールを示しています。

上のペインは、[環境の概要] ページです。環境に関する概要情報が表示されます。

ページの下半分には、より詳細な情報を提供するタブが表示されます。[イベント] タブはデフォルトで表示されます。タブにリンクされているページは、環境の左側のナビゲーションペインにも表示されます。

コンソールのナビゲーションペインには、環境にデプロイされているアプリケーションの名前と、関連するアプリケーション管理ページが表示されます。環境名はナビゲーションページにも表示され、それに続いて環境管理ページがあります。環境名の下にリストされているリンクには、前述のタブ付きページに加えて、[環境に移動] および [設定] も含まれます。

環境の概要のペイン

このトピックでは、[環境の概要] ペインが提供する情報について説明します。環境に関する概要情報が環境管理コンソールの上半分に表示されます。

次の図は、[環境の概要] ペインを示しています。

健康

環境全体の状態。環境のヘルスが悪化すると、環境のヘルスの横に [原因を表示] リンクが表示されます。このリンクを選択すると、[ヘルス] タブに詳細が表示されます。

ドメイン

環境の [Domain] (ドメイン)、つまり URL は、[Environment overview] (環境の概要) ページの上部、環境の [Health] (ヘルス) の下にあります。これは、環境で実行するウェブアプリケーションの URL です。URL を選択してアプリケーションを起動できます。

環境 ID

環境 ID。これは環境の作成時に生成される内部 ID です。

アプリケーション名

環境にデプロイされ、実行されているアプリケーションの名前。

実行バージョン

環境にデプロイされ、実行されているアプリケーションバージョンの名前。Upload and Deploy を選択すると、ソースバンドルをアップロードし、環境にデプロイすることができます。このオプションにより、新しいアプリケーションバージョンが作成されます。

プラットフォーム

その環境で実行されているプラットフォームバージョンの名前です。通常この名前は、アーキテクチャ、オペレーティングシステム (OS)、言語、およびアプリケーションサーバー (総称してプラットフォームブランチと呼ばれます) と、プラットフォームの特定のバージョン番号で構成されます。

プラットフォームバージョンが最新のものではない場合は、[プラットフォーム] セクションの横にステータスラベルが表示されます。[更新] ラベルは、ご利用のプラットフォームバージョンはサポートされているが、より新しいバージョンが利用可能であることを示します。プラットフォームバージョンには、[非推奨] または [廃止] というラベルが付いている場合もあります。[バージョンを変更] を選択して、プラットフォームブランチを新しいバージョンに更新します。プラットフォームバージョンの状態の詳細については、「Elastic Beanstalk プラットフォームの用語集」の「プラットフォームブランチ」セクションを参照してください。このページの前に示した図は、特定のプラットフォームの [更新] ステータスラベルを示しています。

環境の詳細

このトピックでは、環境管理コンソールが左側のナビゲーションペインとタブ付きページから提供する追加情報について説明します。

次の図は、環境管理コンソールを示しています。

環境管理コンソールの下半分には、環境に関するより詳細で多様な情報を提供するタブが一覧表示されます。左側のナビゲーションペインからタブページまたはページラベルを選択できます。

環境名の下のコンソールの左側のナビゲーションペインには、タブ付きページにない 2 つの選択肢があります。それは [環境に移動] と [設定] です。

> **Note**
>
> [環境に移動] を選択してアプリケーションを起動します。

設定

左側のナビゲーションペインの [設定] ページを使用して、環境とそのリソースの現在の設定を表示および更新します。これには、ネットワーク設定、データベース設定、ロードバランシング、通知、ヘルスモニタリング設定、マネージドプラットフォーム更新設定、デプロイ設定、インスタンスログストリーミング、CloudWatch 統合、プロキシサーバー設定 AWS X-Ray、環境プロパティ、プラットフォーム固有のオプションが含まれます。このページの設定を使用して、デプロイ中の環境動作のカスタマイズ、追加機能の有効化、環境作成時に選択したインスタンスタイプおよびその他の設定の修正が可能です。

詳細については、「Elastic Beanstalk 環境の設定」を参照してください。

イベント

[イベント] ページには、環境のイベント ストリームが表示されます。Elastic Beanstalk は、環境の操作時、および操作の結果として環境のリソースが作成あるいは変更された場合に、イベントメッセージを出力します。

詳細については、「Elastic Beanstalk 環境のイベントストリームの表示」を参照してください。

健康

拡張ヘルスモニタリングが有効になっている場合、このページには環境内の EC2 インスタンスと各インスタンスのライブヘルス情報が一覧表示されます。

[全体的なヘルス] ページには、環境のすべてのインスタンスを組み合わせた平均的なヘルスデータが表示されます。

[拡張インスタンスヘルス] ペインには、環境内の個々の EC2 インスタンスのライブヘルス情報が表示されます。拡張ヘルスモニタリングによって、Elastic Beanstalk は環境のリソースをより厳密にモニタリングできるため、アプリケーションのヘルスをより正確に評価できます。

拡張ヘルスモニタリングを有効化すると、このページには環境のインスタンスから送られたリクエストに関する情報や、レイテンシーやロード、CPU 使用率などオペレーティングシステムからのメトリクスが表示されます。

詳細については、「Elastic Beanstalk でのヘルスレポートとモニタリングの強化」を参照してください。

ログ

[Logs] ページにより、環境で EC2 インスタンスからのログを取得できます。ログをリクエストすると、Elastic Beanstalk がインスタンスにコマンドを送信します。これにより、Amazon S3 の Elastic Beanstalk ストレージバケットにログがアップロードされます。このページでログをリクエストすると、15 分後に Elastic Beanstalk がログを自動的に Amazon S3 から削除します。

ローカルでのローテーション後、ログを Amazon S3にアップロードして永続的に保管できるように、環境のインスタンスを設定することもできます。

詳細については、「Elastic Beanstalk 環境の Amazon EC2 インスタンスからのログの表示」を参照してください。

モニタリング

[Monitoring] ページには、環境のヘルス情報の概要が表示されます。これには、Elastic Load Balancing および Amazon EC2 によって提供されるデフォルトのメトリクスセットと、環境の経時変化のグラフが含まれます。

詳細については、「AWS マネジメントコンソールでの環境の状態のモニタリング」を参照してください。

アラーム

[既存のアラーム] ページには、環境用に設定したすべてのアラームに関する情報が表示されます。このページのオプションを使用して、アラームの変更や削除ができます。

詳細については、「アラームの管理」を参照してください。

マネージド更新

[マネージド更新] ページには、今後および完了したマネージドプラットフォーム更新やインスタンスの置換に関する情報が表示されます。

マネージド更新機能により、選択した週ごとのメンテナンス期間で最新のプラットフォームバージョンへ自動的に更新するように、環境を設定することもできます。プラットフォームのリリースが更新される際にも、メンテナンス期間中に Amazon EC2 インスタンスをすべて置換するよう環境を設定することができます。これによって、アプリケーションを長期間にわたり実行した場合に発生する問題を軽減できます。

詳細については、「マネージドプラットフォーム更新」を参照してください。

[タグ]

[タグ] ページには、環境を作成した際に Elastic Beanstalk によって適用されたタグや、独自に追加したタグが表示されます。カスタムタグを追加、編集、削除できます。Elastic Beanstalk によって適用されたタグを編集または削除することはできません。

環境タグは、アプリケーションをサポートするために Elastic Beanstalk が作成したすべてのリソースに適用されます。

詳細については、「Elastic Beanstalk 環境でのリソースのタグ付け」を参照してください。

環境アクション

このトピックでは、環境管理コンソールの [アクション] ドロップダウンメニューから、環境で実行するために選択できる一般的なオペレーションについて説明します。

次の図は、環境管理コンソールを示しています。[アクション] ドロップダウンメニューは、環境名を表示するヘッダーの右側、[更新] ボタンの横にあります。

> **Note**
> 一部のアクションは、特定の条件でのみ使用でき、適切な条件が満たされるまで無効のままです。

設定のロード

以前に保存した設定をロードします。設定はアプリケーションに保存されており、関連付けられた環境にロードできます。環境の設定に変更を加えた場合、保存された設定をロードすることで、それらの変更を元に戻すことができます。同じアプリケーションを実行している別の環境から保存した設定をロードして、環境間で設定を伝達することができます。

設定の保存

環境の現在の設定をアプリケーションに保存します。環境の設定に変更を加える前に、必要に応じて後でロールバックできるように、現在の設定を保存します。新しい環境を起動するときに、保存された設定を適用することもできます。

環境ドメイン (URL) のスワップ

現在の環境の CNAME を新しい環境とスワップします。CNAME をスワップした後は、その環境の URL を使用するアプリケーションへのすべてのトラフィックが新しい環境に送られます。新しいバージョンのアプリケーションをデプロイする準備ができたら、新しいバージョンで別の環境を起動できます。新しい環境でリクエストを受け取る準備が完了したら、CNAME スワップを実行して、その環境へのトラフィックのルーティングを開始します。これを実行してもサービスが中断されることはありません。詳細については、「Elastic Beanstalk を使用したブルー/グリーンデプロイ」を参照してください。

環境のクローンを作成する

現在実行中の環境と同じ設定で、新しい環境を起動します。

最新のプラットフォームを使用してクローンを作成する

使用中の Elastic Beanstalk プラットフォームの最新バージョンを使用して、現在の環境のクローンを作成します。このオプションは、現在の環境のプラットフォームについて新しいバージョンが利用可能な場合にのみ使用できます。

現在のオペレーションを中止する

進行中の環境の更新を停止します。オペレーションを中止すると、その処理の進行度合いによって、環境内のインスタンスの一部が他のインスタンスと異なる状態になる場合があります。このオプションは、環境が更新中である場合にのみ使用できます。

アプリケーションサーバーを再起動する

環境内のインスタンスで実行されているウェブサーバーを再起動します。このオプションは、 AWS リソースを終了または再起動しません。無効なリクエストへの応答中に環境の動作がおかしくなった場合、根本原因のトラブルシューティングを行う間、アプリケーションサーバーを再起動することで、機能を一時的に回復できます。

環境の再構築

実行中の環境のすべてのリソースを終了し、同じ設定で新しい環境を構築します。新しい環境を最初からデプロイするのと同様に、この処理の完了には数分が必要となります。環境のデータ層で実行されている Amazon RDS インスタンスがある場合は、すべて再構築時に削除されます。そのデータが必要な場合は、スナップショットを作成します。RDS コンソール上で手動でスナップショットを作成するか、データ層の削除ポリシーを設定して、インスタンスを削除する前にスナップショットを自動的に作成させることができます。データ層を作成した際は、デフォルトでこの設定になっています。

環境の終了

実行中の環境ですべてのリソースを終了した上で、アプリケーションからその環境を削除します。データ層で実行されている RDS インスタンスがあり、その中のデータを保存しておく必要がある場合は、データベース削除ポリシーを `Snapshot` または `Retain` に設定します。詳細については、このガイドの環境の設定の章で「データベースのライフサイクル」を参照してください。

Elastic Beanstalk 環境の作成

次の手順は、デフォルトアプリケーションを実行する新しい環境を起動します。これらのステップは、デフォルトのオプション値を使用して、環境を迅速に起動して実行できるように単純化されています。

> **アクセス許可に関する注意**
>
> 環境を作成するには、Elastic Beanstalk フルアクセス管理ポリシーのアクセス許可が必要です。詳細については、「Elastic Beanstalk ユーザーポリシー」を参照してください。

サンプルアプリケーションを使用して環境を起動するには (コンソール)

1. Elastic Beanstalk 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、[アプリケーション] を選択します。リストから既存のアプリケーションを選択します。 アプリケーションの管理 の手順に従って作成することもできます。
3. アプリケーションの概要ページで、[Create new environment] (新しい環境の作成) を選択します。

   次の図は、アプリケーションの概要ページを示しています。

   これにより、[Create environment] (環境を作成する) ウィザードが起動します。ウィザードには、新しい環境を作成するための一連のステップが用意されています。
4. [環境枠] では、[ウェブサーバー環境] または [ワーカー環境] の環境枠を選択します。作成後に環境枠を変更することはできません。

   > **Note**
   >
   > Windows Server プラットフォームの .NET はワーカー環境枠をsupport していません。

   [アプリケーション情報] フィールドは、以前に選択したアプリケーションに基づいてデフォルトで設定されます。

   [環境情報] では、アプリケーション名に基づいて [環境名] がデフォルトでグループ化されます。別の環境名を使用する場合は、フィールドに別の値を入力できます。必要に応じて [ドメイン] の名前を入力できます。入力しない場合、Elastic Beanstalk は値を自動的に生成します。必要に応じて [環境の説明] を入力することもできます。
5. プラットフォーム では、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。

> **Note**
>
> Elastic Beanstalk では、一覧表示されるほとんどのプラットフォームで複数のバージョンがサポートされています。デフォルトでは、選択したプラットフォームとプラットフォームブランチの推奨バージョンがコンソールによって選択されます。アプリケーションで異なるバージョンが必要な場合は、ここでそのバージョンを選択できます。サポートされているプラットフォームのバージョンについては、the section called “サポートされているプラットフォーム” を参照してください。

6. [アプリケーションコード] では、サンプルアプリケーションを起動するためのいくつかの選択肢があります。
   - ソースコードを指定せずにデフォルトのサンプルアプリケーションを起動するには、[サンプルアプリケーション] を選択します。このアクションは、以前に選択したプラットフォームに対して Elastic Beanstalk が提供する単一ページアプリケーションを選択します。
   - このガイドまたは他のソースからサンプルアプリケーションをダウンロードした場合は、次の手順を実行します。
     a. [コードのアップロード] を選択します。
     b. 次に [ローカルファイル] を選択し、[アプリケーションをアップロード] で [ファイルを選択] を選択します。
     c. コンピュータのオペレーティングシステムには、ダウンロードしたローカルファイルを選択するためのインターフェイスが表示されます。ソースバンドルファイルを選択して続行します。
7. [プリセット] では、[単一インスタンス] を選択します。
8. [次へ] を選択します。
9. [サービスアクセスの設定] ページが表示されます。

   次の図は、[サービスアクセスの設定] ページを示しています。

10. [既存のサービスロール] ドロップダウンから値を選択します。
11. (オプション) EC2 キーペアを以前に作成している場合は、[EC2 キーペア] フィールドドロップダウンから選択できます。これを使用して、Elastic Beanstalk がアプリケーション用にプロビジョニングした Amazon EC2 インスタンスに安全にログインできます。このステップをスキップした場合、環境の作成後に EC2 キーペアをいつでも作成して割り当てることができます。詳細については、「EC2 キーペア」を参照してください。

12. 次に、[EC2 インスタンスプロファイル] ドロップダウンリストに焦点を当てます。このドロップダウンリストに表示される値は、アカウントが以前に新しい環境を作成したかどうかによって異なる場合があります。

    リストに表示されている値に基づいて、次のいずれかの項目を選択します。

    - `aws-elasticbeanstalk-ec2-role` がドロップダウンリストに表示されている場合は、それをドロップダウンリストから選択します。
    - リストに別の値が表示され、かつそれが環境向けのデフォルト EC2 インスタンスプロファイルである場合、その値をドロップダウンリストから選択します。
    - [EC2 インスタンスプロファイル] ドロップダウンリストに値が表示されない場合は、インスタンスプロファイルを作成する必要があります。

    > **インスタンスプロファイルを作成する**
    >
    > インスタンスプロファイルを作成するには、同じページの別の手順に迂回します。この手順の末尾に移動し、次の手順である「EC2 インスタンスプロファイル の IAM ロールを作成する」を展開します。
    > 「EC2 インスタンスプロファイルの IAM ロールを作成する」ステップを完了し、その後 [EC2 インスタンスプロファイル] 用に選択できる IAM ロールを作成します。その後、このステップに戻ります。

    IAM ロールを作成してリストを更新すると、ドロップダウンリストに選択肢として表示されます。[EC2 インスタンスプロファイル] ドロップダウンリストから、先ほど作成した IAM ロールを選択します。

13. [Configure service access] (サービスアクセスの設定) ページで [Skip to Review] (確認をスキップ) を選択します。

    これにより、このステップのデフォルト値が選択され、オプションのステップはスキップされます。

14. [Review] (レビュー) ページに、すべての選択内容の概要が表示されます。

    環境をさらにカスタマイズするには、設定する項目を含むステップの横にある [Edit] (編集) を選択します。以下のオプションは、環境の作成中にのみ設定できます。

    - 環境名

- ドメイン名
- プラットフォームのバージョニング
- プロセッサ
- VPC
- 階層

次の設定は環境の作成後に変更できますが、新しいインスタンスあるいはその他のリソースをプロビジョニングする必要があり、適用までに長い時間がかかる場合があります。

- インスタンスタイプ、ルートボリューム、キーペア、および AWS Identity and Access Management (IAM) ロール
- 内部 Amazon RDS データベース
- ロードバランサー

すべての使用できる設定の詳細については、「新しい環境の作成ウィザード」を参照してください。

15. ページ下部の [Submit] (送信) を選択して、新しい環境の作成を開始します。

EC2 インスタンスプロファイルの IAM ロールを作成

EC2 インスタンスプロファイルを作成するには

1. [ロールの作成] を選択してください。
2. [信頼できるエンティティタイプ] で、[AWS サービス] を選択します。
3. ユースケースでは、Elastic Beanstalk – Compute を選択します。
4. [次へ] を選択します。
5. アクセス許可ポリシーに以下が含まれていることを確認し、次へを選択します。
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`
6. [ロールの作成] を選択してください。

7. サービスアクセスの設定タブに戻り、リストを更新して、新しく作成した EC2 インスタンスプロファイルを選択します。

Elastic Beanstalk が環境を作成する間、ユーザーは Elastic Beanstalk コンソールにリダイレクトされます。環境の状態が緑色に切り替わったら、環境名の横の URL を選択して実行中のアプリケーションを表示します。この URL は、内部ロードバランサーでカスタム VPC を使用するよう環境を設定しない限り、通常はインターネットからアクセス可能です。

トピック

- 新しい環境の作成ウィザード
- Elastic Beanstalk 環境のクローンを作成する
- Elastic Beanstalk 環境を終了する
- AWS CLI を使用した Elastic Beanstalk 環境の作成
- API を使用した Elastic Beanstalk 環境の作成
- 「Launch Now URL」の作成
- Elastic Beanstalk 環境のグループを作成および更新する

新しい環境の作成ウィザード

このトピックでは、[Create environment] (環境の作成) ウィザードの詳細と、作成する環境を設定するために使用できるすべての方法について説明します。

> **Note**
>
> Elastic Beanstalk 環境の作成 では、[環境の作成] ウィザードを起動し、デフォルト値と推奨設定を使用して環境をすばやく作成する方法について説明します。このトピックでは、すべてのオプションについて説明します。

ウィザードページ

[Create environment] (環境の作成) ウィザードには、新しい環境を作成するための一連の手順が用意されています。

環境枠

環境枠では、[Web server environment] (ウェブサーバー環境) または [Worker environment] (ワーカー環境) の環境枠を選択します。作成後に環境枠を変更することはできません。

> **Note**
>
> Windows Server プラットフォームの .NET はワーカー環境枠をsupport していません。

アプリケーション情報

[Application overview] (アプリケーションの概要) ページから [Create new environment] (新しい環境の作成) を選択してウィザードを起動した場合、[Application name] (アプリケーション名) は事前に入力されています。それ以外の場合は、アプリケーション名を入力します。必要に応じて、アプリケーションタグを追加します。

環境情報

環境の名前とドメインを設定し、環境の説明を作成します。これらの環境設定は、環境の作成後に変更できないことに注意してください。

- [名前] – 環境の名前を入力します。フォームで名前が生成されます。
- [ドメイン] – (ウェブサーバー環境) 環境の一意なドメイン名を入力します。デフォルトの名前は環境の名前です。別のドメイン名を入力することができます。Elastic Beanstalk は、この名前を使用して、環境を示す一意の CNAME を作成します。目的のドメイン名が利用可能かどうか確認するには、[Check Availability] を選択します。
- [説明] – この環境の説明を入力します。

新しい環境に適したプラットフォームを選択する

新しい環境は、次の 2 種類のプラットフォームから作成できます。

- 管理されたプラットフォーム
- カスタムプラットフォーム

管理されたプラットフォーム

ほとんどの場合、新しい環境には Elastic Beanstalk マネージドプラットフォームを使用します。新しい環境ウィザードが起動すると、デフォルトで [管理されたプラットフォーム] オプションが選択されます。

プラットフォーム、プラットフォーム内のプラットフォームブランチ、およびブランチ内の特定のプラットフォームバージョンを選択します。プラットフォームブランチを選択すると、ブランチ内の推奨バージョンがデフォルトで選択されます。以前に使用したことがあるプラットフォームのバージョンを選択することもできます。

> **Note**
>
> 本番環境では、サポートされているプラットフォームブランチのプラットフォームバージョンを選択することをお勧めします。プラットフォームブランチの状態についての詳細は、the section called “プラットフォームの用語集”にあるプラットフォームブランチの定義を参照してください。

カスタムプラットフォーム

既製のプラットフォームがニーズに合わない場合は、カスタムプラットフォームから新しい環境を作成することができます。カスタムプラットフォームを指定するには、[カスタムプラットフォーム] オプションを選択し、使用可能なカスタムプラットフォームのいずれかを選択します。利用可能なカスタムプラットフォームがない場合、このオプションはグレー表示されます。

アプリケーションコードを提供する

使用するプラットフォームを選択したので、次のステップではアプリケーションコードを提供します。

これには複数のオプションがあります。

- Elastic Beanstalk が各プラットフォーム用に提供するサンプルアプリケーションを使用できます。
- 既に Elastic Beanstalk にデプロイしたコードを使用できます。[Existing version] セクションを選択します。アプリケーションは [Application code] セクションにあります。
- 新しいコードをアップロードできます。[コードのアップロード]、[アップロード] の順に選択します。ローカルファイルから新しいアプリケーションコードをアップロードすることも、アプリケーションコードを含む Amazon S3 バケットの URL を指定することもできます。

> **Note**
>
> 選択したプラットフォームのバージョンに応じて、ZIP ソースバンドル、WAR ファイル、またはプレーンテキストの Docker 設定にアプリケーションをアップロードできます。ファイルサイズの上限は 500 MB です。

また、新しいコードのアップロードを選択すると、コードに関連付けるタグを指定することもできます。アプリケーションバージョンのタグ付けの詳細については、「the section called “アプリケーションバージョンのタグ付け”」を参照してください。

デフォルトの設定オプションを使用して環境をすばやく作成するために、[環境の作成] を選択できるようになりました。次のセクションで説明するように、[Configure more options] を選択してさらに設定を変更できます。

ウィザードの設定ページ

[さらにオプションを設定] を選択すると、ウィザードに [設定] ページが表示されます。このページでは、設定プリセットの選択、環境で使用するプラットフォームバージョンの変更、または新しい環境に対する特定の設定を選択できます。

プリセット設定の選択

このページの [プリセット] セクションでは、Elastic Beanstalk の複数の設定プリセットが提供されており、さまざまなユースケースに対応できます。各プリセットには、複数の設定オプション用に推奨値が含まれています。

[High availability (高可用性)] プリセットにはロードバランサーが含まれており、本番稼働環境に推奨されます。高可用性のための複数のインスタンスを実行し、負荷に応じてスケールできる環境が必要な場合は、これを選択します。[Single instance (単一インスタンス)] プリセットは、主に開発用に推奨されます。2 つのプリセットでスポットインスタンスリクエストが有効になります。Elastic Beanstalk のキャパシティー設定の詳細については、「Auto Scaling グループ」を参照してください。

最後のプリセットである [Custom configuration (カスタム設定)] では、ロール設定を除くすべての推奨値を削除し、API デフォルトを使用します。設定オプションを指定する設定ファイルでソースバン

ドルをデプロイしている場合は、このオプションを選択します。[Low cost] または [High availability] 設定プリセットを変更すると、[Custom configuration] も自動的に選択されます。

設定をカスタマイズする

設定プリセットを選択することに加えて (またはその代わりに)、お使いの環境で 設定オプションを微調整することができます。[設定] ウィザードには、複数の設定カテゴリが表示されます。各設定カテゴリには、構成設定のグループの値の要約が表示されます。このグループの設定を編集するには、[編集] を選択します。

設定カテゴリ

- ソフトウェアの設定
- インスタンス
- 容量
- ロードバランサー
- ローリング更新とデプロイ
- セキュリティ
- モニタリング
- マネージド更新
- 通知
- ネットワーク
- データベース
- [タグ]
- ワーカー環境

ソフトウェアの設定

[ソフトウェア設定の変更] ページを使用すると、アプリケーションを実行する Amazon Elastic Compute Cloud (Amazon EC2) インスタンスでソフトウェアを設定できます。環境プロパティ、AWS X-Ray デバッグ、インスタンスログの保存とストリーミング、プラットフォーム固有の設定を設定できます。詳細については、「the section called “環境変数とソフトウェア設定”」を参照してください。

インスタンス

[インスタンスの変更] 設定ページを使用すると、アプリケーションを実行する Amazon EC2 インスタンスを設定できます。詳細については、「the section called “Amazon EC2 インスタンス”」を参照してください。

容量

[容量の変更] 設定ページを使用して、環境のコンピューティング性能と [Auto Scaling グループ] 設定を構成し、使用しているインスタンスの数とタイプを最適化します。また、トリガーまたはスケジュールに基づいて環境の容量を変更することもできます。

負荷分散された環境では、高可用性のための複数のインスタンスを実行し、設定の更新およびデプロイ中にダウンタイムを防ぐことができます。負荷分散された環境では、ドメイン名はロードバランサーにマッピングされます。シングルインスタンス環境では、インスタンスの Elastic IP アドレスにマッピングされます。

> **Warning**
>
> シングルインスタンス環境は、本稼働環境では利用できません。インスタンスがデプロイ中に不安定になった場合、または設定の更新中に Elastic Beanstalk がインスタンスを終了して再起動した場合は、アプリケーションを一定期間使用できなくなることがあります。開発、テスト、またはステージングには、単一インスタンス環境を使用します。本稼働用には負荷分散された環境を使用します。

環境容量の設定についての詳細は、「the section called “Auto Scaling グループ”と the section called “Amazon EC2 インスタンス”」を参照してください。

ロードバランサー

[ロードバランサーの変更] の設定ページを使用して、ロードバランサータイプを選択し、設定を行います。ロードバランサー環境では、環境のロードバランサーがアプリケーションに送信されるすべてのトラフィック用のエントリポイントとなります。Elastic Beanstalk は、いくつかのタイプのロードバランサーをサポートしています。デフォルトでは、Elastic Beanstalk コンソールで Application Load Balancer を作成し、ポート 80 で HTTP トラフィックを処理するように設定します。

> **Note**
> 環境の作成時にのみ、環境のロードバランサーの種類を選択できます。

ロードバランサーの種類と設定の詳細については、「the section called “ロードバランサー”」と「the section called “HTTPS”」を参照してください。

> **Note**
> [環境作成] コンソールウィザードでは、Classic Load Balancer (CLB) オプションは無効になっています。既存の環境を Classic Load Balancer で設定している場合は、Elastic Beanstalk コンソールまたは EB CLI のいずれかを使用して既存の環境を複製することで新しい環境を作成できます。また、EB CLI または を使用して Classic Load Balancer で設定された新しい環境AWS CLIを作成することもできます。これらのコマンドラインツールは、アカウントに CLB がまだ存在しない場合でも、CLB を使用して新しい環境を作成します。

ローリング更新とデプロイ

[ローリング更新とデプロイの変更] 設定ページを使用すると、Elastic Beanstalk が環境のアプリケーションデプロイと設定の更新を処理する方法を設定できます。

アプリケーションのデプロイは、更新されたアプリケーションソースバンドルをアップロードして環境にデプロイするときに発生します。デプロイの設定の詳細については、「the section called “デプロイオプション”」を参照してください。

起動設定または VPC 設定を変更する設定の変更では、環境内のすべてのインスタンスを終了し置き換える必要があります。更新タイプおよびその他のオプションの設定の詳細については、「the section called “設定変更”」を参照してください。

セキュリティ

[Configure service access] (サービスアクセスの設定) ページを使用して、サービスおよびインスタンスのセキュリティ設定を設定します。

Elastic Beanstalk のセキュリティの概念については、「アクセス許可」を参照してください。

Elastic Beanstalk コンソールで初めて環境を作成する場合、デフォルトの許可のセットで EC2 インスタンスプロファイルを作成する必要があります。[EC2 インスタンスプロファイル] ドロップダウンリストに選択できる値が表示されない場合は、以下の手順を拡張してください。後で [EC2 インスタンスプロファイル] に選択できるロールを作成するステップが記載されています。

EC2 インスタンスプロファイルの IAM ロールを作成

EC2 インスタンスプロファイルを作成するには

1. [ロールの作成] を選択してください。
2. [信頼できるエンティティタイプ] で、[AWS サービス] を選択します。
3. ユースケースでは、Elastic Beanstalk – Compute を選択します。
4. [次へ] を選択します。
5. アクセス許可ポリシーに以下が含まれていることを確認し、次へを選択します。
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`
6. [ロールの作成] を選択してください。
7. サービスアクセスの設定タブに戻り、リストを更新して、新しく作成した EC2 インスタンスプロファイルを選択します。

モニタリング

[Modify monitoring (モニタリングの変更)] 設定ページを使用して、ヘルスレポート、モニタリングルール、およびヘルスイベントのストリーミングを設定します。詳細については、「the section called “拡張ヘルスレポートの有効化”、the section called “拡張ヘルスルール”」、および「the section called “環境ヘルスのストリーミング”」を参照してください。

マネージド更新

[Modify managed updates (管理された更新の変更)] 設定ページを使用して、管理されたプラットフォームの更新を設定します。有効にするかどうかの決定や、スケジュールの設定、およびその他のプロパティを設定できます。詳細については、「the section called “マネージド更新”」を参照してください。

通知

[Modify notifications (通知の変更)] 設定ページを使用して、環境から重要なイベントの E メール通知を受信する E メールアドレスを指定します。

ネットワーク

カスタム VPC を作成した場合、[Modify network (ネットワークの変更)] 設定ページで、作成した VPC を使用するように環境を設定します。VPC を選択しなかった場合は、デフォルトの VPC とサブネットが Elastic Beanstalk で使用されます。

データベース

[データベースの変更] ページを使用すると、開発またはテスト用に、Amazon Relational Database Service (Amazon RDS) データベースを環境に追加できます。Elastic Beanstalk はデータベースのホスト名、ユーザー名、パスワード、テーブル名、ポートの環境プロパティを設定して、インスタンスに接続情報を提供します。

詳細については、「the section called “データベース”」を参照してください。

[タグ]

[タグの変更] 設定ページを使用して、環境内のリソースにタグを追加します。環境タグ付けの詳細については、「Elastic Beanstalk 環境でのリソースのタグ付け」を参照してください。

ワーカー環境

ワーカー枠環境を作成する場合は、[Modify worker (ワーカーの変更)] 設定ページを使用してワーカー環境を設定します。環境内のインスタンスのワーカーデーモンは、Amazon Simple Queue Service (Amazon SQS) キューから項目を取り出し、ポストメッセージとしてワーカーアプリケーションに中継します。ワーカーデーモンが読み取る Amazon SQS キュー (自動生成または既存) を選択できます。ワーカーデーモンがアプリケーションに送信するメッセージを設定することもできます。

詳細については、「」を参照してくださいthe section called “ワーカー環境”

Elastic Beanstalk 環境のクローンを作成する

既存の Elastic Beanstalk 環境のクローンを作成することで、新しい環境の基盤として既存の環境を利用することができます。たとえば、元の環境のプラットフォームで使用されていたプラットフォームブランチの新しいバージョンを使用するためにクローンを作成する場合があります。Elastic Beanstalk によって、元の環境で使用されていた環境設定がクローンに適用されます。新しい環境を作成する代わりに既存の環境をクローン化することで、Elastic Beanstalk サービスに適用していたオプション設定や環境変数などの設定を手動で行う必要がなくなります。Elastic Beanstalk は、元の環境に関連付けられた AWS リソースのコピーも作成します。

以下の状況に注意することが重要です。

- クローン作成プロセスでは、Elastic Beanstalk によって Amazon RDS からクローンにデータがコピーされることはありません。
- Elastic Beanstalk では、リソースに対するアンマネージド型の変更はクローンに含められません。Elastic Beanstalk コンソール、コマンドラインツール、API 以外のツールを使用して AWS リソースに加えた変更は、アンマネージド型の変更と見なされます。
- イングレスのセキュリティグループは、アンマネージド型の変更と見なされます。クローンの Elastic Beanstalk 環境は、イングレス用のセキュリティグループを引き継ぐのではなく、環境はすべてのインターネットトラフィックに対してオープンのままになります。クローンの環境のイングレスセキュリティグループを再確立する必要があります。

同じプラットフォームブランチの異なるプラットフォームバージョンにのみ、環境のクローンを作成できます。別のプラットフォームブランチに互換性があるとは限りません。別のプラットフォームブランチを使用するには、新しい環境を手動で作成し、アプリケーションコードをデプロイします。次に、新しいプラットフォームブランチでアプリケーションが正しく動作するように、コードとオプションを必要に応じて変更する必要があります。

AWS マネジメントコンソール

> **Important**
>
> クローンの Elastic Beanstalk 環境は、イングレス用のセキュリティグループを引き継ぐのではなく、環境はすべてのインターネットトラフィックに対してオープンのままになります。クローンの環境のイングレスセキュリティグループを再確立する必要があります。

環境設定のドリフトステータスを確認することで、クローン化できないリソースを確認できます。詳細については、「AWS CloudFormation ユーザーガイド」の「CloudFormation スタック全体のドリフトを検出する」を参照してください 。

環境のクローンを作成するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境概要ページで [アクション] を選択します。
4. [クローン環境] を選択します。
5. [環境のクローンの作成] ページの [元の環境] セクションの情報を見て、クローンを作成したい環境が選択されていることを確認します。
6. [New Environment] (新しい環境) セクションでは、Elastic Beanstalk によって元の環境に基づいて自動的に設定される [Environment name] (環境名)、[Environment URL] (環境 URL)、[Description] (説明)、[Platform version] (プラットフォームのバージョン)、および [Service role] (サービスロール) の各値も必要に応じて変更できます。

   > **Note**
   >
   > 元の環境で使用されているプラットフォームのバージョンが、プラットフォームブランチでの使用が推奨されていない場合には、別のプラットフォームバージョンが推奨されるという警告が表示されます。[プラットフォームのバージョン] を選択すると、推奨されるプラットフォームのバージョンが [3.3.2 (推奨)] のようにリストに表示されます。

7. 準備ができたら、[Clone] を選択します。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI)

> **Important**
>
> クローンの Elastic Beanstalk 環境は、イングレス用のセキュリティグループを引き継ぐのではなく、環境はすべてのインターネットトラフィックに対してオープンのままになります。クローンの環境のイングレスセキュリティグループを再確立する必要があります。

環境設定のドリフトステータスを確認することで、クローン化できないリソースを確認できます。詳細については、「AWS CloudFormation ユーザーガイド」の「CloudFormation スタック全体のドリフトを検出する」を参照してください 。

次に示しように、eb clone コマンドを使用し、実行中の環境をクローニングします。

```
~/workspace/my-app$ eb clone my-env1
Enter name for Environment Clone
(default is my-env1-clone): my-env2
Enter DNS CNAME prefix
(default is my-env1-clone): my-env2
```

クローンコマンドでは、ソース環境の名前を指定するか、指定せずに現在のプロジェクトフォルダのデフォルト環境をクローニングすることができます。EB CLI は、新しい環境の名前と DNS プレフィックスの入力を求めるプロンプトを表示します。

デフォルトでは、eb clone はソース環境のプラットフォームの使用可能な最新バージョンで新しい環境を作成します。新しいバージョンが使用可能であっても同じバージョンを使用するように EB CLI に強制するには、`--exact` オプションを使用します。

```
~/workspace/my-app$ eb clone --exact
```

このコマンドの詳細については、「eb clone」を参照してください。

Elastic Beanstalk 環境を終了する

Elastic Beanstalk コンソールを使用して、実行中の AWS Elastic Beanstalk 環境を終了できます。これにより、未使用の AWS リソースに対して料金が発生するのを回避できます。

> **Note**
> いつでも、また同じバージョンを使用して新しい環境を起動できます。

保存する必要のある環境のデータがある場合、環境を終了する前にデータベース削除ポリシーを `Retain` に設定します。これにより、データベースは Elastic Beanstalk の外部で動作し続けます。この後、Elastic Beanstalk 環境は外部データベースとして接続する必要があります。データベースを動作させずにデータをバックアップする場合は、環境を終了する前にデータベースのスナップショッ

トを作成するように削除ポリシーを設定します。詳細については、このガイドの環境の設定の章で「データベースのライフサイクル」を参照してください。

Elastic Beanstalk は環境の終了に失敗する場合があります。一般的な原因の 1 つとして、別の環境のセキュリティグループが、終了しようとしている環境のセキュリティグループに依存している場合があります。この問題を回避する方法については、このガイドの [EC2 インスタンス] ページで「EC2 セキュリティグループ」を参照してください。

> **Important**
>
> 環境を終了する場合は、作成した CNAME マッピングも削除する必要があります。これにより、使用可能になったホスト名を他のお客様が再利用できます。DNS エントリのダングリングを防ぐため、終了した環境を指す DNS レコードを必ず削除してください。DNS エントリがダングリングしていると、ユーザーのドメイン宛のインターネットトラフィックがセキュリティの脆弱性にさらされる可能性があります。また、他のリスクをもたらす可能性もあります。
>
> 詳細については、Amazon Route 53 デベロッパーガイドの「Route 53 でのダングリング委任レコードからの保護」を参照してください。また、ダングリング DNS エントリの詳細については、AWS セキュリティブログの 「Enhanced Domain Protections for Amazon CloudFront Requests」を参照してください。

Elastic Beanstalk コンソール

環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

> **Note**
>
> 環境を終了すると、終了した環境に関連付けられていた CNAME は解放され、すべてのユーザーが使用できるようになります。

Elastic Beanstalk が環境で実行されている AWS リソースを終了するまでに数分かかります。

AWS CLI

環境を終了するには

- 以下のコマンドを実行してください。

```
$ aws elasticbeanstalk terminate-environment --environment-name my-env
```

API

環境を終了するには

- 以下のパラメータを使って TerminateEnvironment を呼び出します。

  EnvironmentName = SampleAppEnv

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&Operation=TerminateEnvironment
&AuthParams
```

AWS CLI を使用した Elastic Beanstalk 環境の作成

Elastic Beanstalk の AWS CLI コマンドの詳細については、AWS CLI 「コマンドリファレンス」を参照してください。

1. 環境の CNAME が使用可能かどうかを確認します。

```
$ aws elasticbeanstalk check-dns-availability --cname-prefix my-cname
{
    "Available": true,
    "FullyQualifiedCNAME": "my-cname.elasticbeanstalk.com"
}
```

2. アプリケーションバージョンが存在することを確認します。

```
$ aws elasticbeanstalk describe-application-versions --application-name my-app --
version-label v1
```

ソースのアプリケーションバージョンがない場合は、作成します。たとえば、次のコマンドでは、Amazon Simple Storage Service (Amazon S3) のソースバンドルからアプリケーションバージョンを作成できます。

```
$ aws elasticbeanstalk create-application-version --application-name my-app --
version-label v1 --source-bundle S3Bucket=amzn-s3-demo-bucket,S3Key=my-source-
bundle.zip
```

3. アプリケーションの設定テンプレートを作成します。

```
$ aws elasticbeanstalk create-configuration-template --application-name my-app --
template-name v1 --solution-stack-name "64bit Amazon Linux 2015.03 v2.0.0 running
 Ruby 2.2 (Passenger Standalone)"
```

4. 環境を作成します。

```
$ aws elasticbeanstalk create-environment --cname-prefix my-cname --application-
name my-app --template-name v1 --version-label v1 --environment-name v1clone --
option-settings file://options.txt
```

オプション設定は、options.txt ファイルで定義されます。

```
[
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "IamInstanceProfile",
        "Value": "aws-elasticbeanstalk-ec2-role"
    }
]
```

上記のオプション設定は、IAM インスタンスプロファイルを定義します。ARN またはプロファイル名を指定します。

5. 新しい環境が緑色で表示され、準備が完了していることを確認します。

```
$ aws elasticbeanstalk describe-environments --environment-names my-env
```

新しい環境が緑色で表示されておらず、準備も完了していない場合は、操作をやり直すか、または環境をそのままにして調査を行う必要があります。操作が終わったら環境を終了し、使用していないリソースをすべてクリーンアップします。

> **Note**
> 環境の起動に時間がかかる場合は、タイムアウト期限を調整することができます。

API を使用した Elastic Beanstalk 環境の作成

1. 以下のパラメータを使って `CheckDNSAvailability` を呼び出します。

   - `CNAMEPrefix` = `SampleApp`

   Example

   ```
   https://elasticbeanstalk.us-east-2.amazonaws.com/?CNAMEPrefix=sampleapplication
   &Operation=CheckDNSAvailability
   &AuthParams
   ```

2. 以下のパラメータを使用して、`DescribeApplicationVersions` を呼び出します。

   - `ApplicationName` = `SampleApp`
   - `VersionLabel` = `Version2`

   Example

   ```
   https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
   &VersionLabel=Version2
   &Operation=DescribeApplicationVersions
   &AuthParams
   ```

3. 以下のパラメータを使用して、`CreateConfigurationTemplate` を呼び出します。

   - `ApplicationName` = `SampleApp`
   - `TemplateName` = `MyConfigTemplate`
   - `SolutionStackName` = `64bit%20Amazon%20Linux%202015.03%20v2.0.0%20running%20Ruby%202.2%20(Passenger%20Standalone)`

Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
&TemplateName=MyConfigTemplate
&Operation=CreateConfigurationTemplate
&SolutionStackName=64bit%20Amazon%20Linux%202015.03%20v2.0.0%20running%20Ruby
%202.2%20(Passenger%20Standalone)
&AuthParams
```

4. 以下のパラメータセットのいずれかで `CreateEnvironment` を呼び出します。

   a. ウェブサーバー環境枠に対しては次を使用します:

   - `EnvironmentName` = `SampleAppEnv2`
   - `VersionLabel` = `Version2`
   - `Description` = `description`
   - `TemplateName` = `MyConfigTemplate`
   - `ApplicationName` = `SampleApp`
   - `CNAMEPrefix` = `sampleapplication`
   - `OptionSettings.member.1.Namespace` = `aws:autoscaling:launchconfiguration`
   - `OptionSettings.member.1.OptionName` = `IamInstanceProfile`
   - `OptionSettings.member.1.Value` = `aws-elasticbeanstalk-ec2-role`

   Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&TemplateName=MyConfigTemplate
&CNAMEPrefix=sampleapplication
&Description=description
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=aws-elasticbeanstalk-ec2-role
```

```
&AuthParams
```

b. ワーカー環境枠に対しては次を使用します:

- EnvironmentName = SampleAppEnv2
- VersionLabel = Version2
- Description = description
- TemplateName = MyConfigTemplate
- ApplicationName = SampleApp
- Tier = Worker
- OptionSettings.member.1.Namespace = aws:autoscaling:launchconfiguration
- OptionSettings.member.1.OptionName = IamInstanceProfile
- OptionSettings.member.1.Value = aws-elasticbeanstalk-ec2-role
- OptionSettings.member.2.Namespace = aws:elasticbeanstalk:sqsd
- OptionSettings.member.2.OptionName = WorkerQueueURL
- OptionSettings.member.2.Value = sqsd.elasticbeanstalk.us-east-2.amazonaws.com
- OptionSettings.member.3.Namespace = aws:elasticbeanstalk:sqsd
- OptionSettings.member.3.OptionName = HttpPath
- OptionSettings.member.3.Value = /
- OptionSettings.member.4.Namespace = aws:elasticbeanstalk:sqsd
- OptionSettings.member.4.OptionName = MimeType
- OptionSettings.member.4.Value = application/json
- OptionSettings.member.5.Namespace = aws:elasticbeanstalk:sqsd
- OptionSettings.member.5.OptionName = HttpConnections
- OptionSettings.member.5.Value = 75
- OptionSettings.member.6.Namespace = aws:elasticbeanstalk:sqsd
- OptionSettings.member.6.OptionName = ConnectTimeout
- OptionSettings.member.6.Value = 10
- OptionSettings.member.7.Namespace = aws:elasticbeanstalk:sqsd
- OptionSettings.member.7.OptionName = InactivityTimeout

- OptionSettings.member.7.Value = 10
- OptionSettings.member.8.Namespace = aws:elasticbeanstalk:sqsd
- OptionSettings.member.8.OptionName = VisibilityTimeout
- OptionSettings.member.8.Value = 60
- OptionSettings.member.9.Namespace = aws:elasticbeanstalk:sqsd
- OptionSettings.member.9.OptionName = RetentionPeriod
- OptionSettings.member.9.Value = 345600

Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&TemplateName=MyConfigTemplate
&Description=description
&Tier=Worker
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=aws-elasticbeanstalk-ec2-role
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.2.OptionName=WorkerQueueURL
&OptionSettings.member.2.Value=sqsd.elasticbeanstalk.us-east-2.amazonaws.com
&OptionSettings.member.3.Namespace=aws%3elasticbeanstalk%3sqsd
&OptionSettings.member.3.OptionName=HttpPath
&OptionSettings.member.3.Value=%2F
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.4.OptionName=MimeType
&OptionSettings.member.4.Value=application%2Fjson
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.5.OptionName=HttpConnections
&OptionSettings.member.5.Value=75
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.6.OptionName=ConnectTimeout
&OptionSettings.member.6.Value=10
&OptionSettings.member.7.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.7.OptionName=InactivityTimeout
&OptionSettings.member.7.Value=10
&OptionSettings.member.8.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.8.OptionName=VisibilityTimeout
```

```
&OptionSettings.member.8.Value=60
&OptionSettings.member.9.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.9.OptionName=RetentionPeriod
&OptionSettings.member.9.Value=345600
&AuthParams
```

「Launch Now URL」の作成

誰でも迅速に事前定義されたウェブアプリケーションを AWS Elastic Beanstalkにデプロイして実行できるように、カスタム URL を作成できます。この URL は「Launch Now URL」と呼ばれます。「Launch Now URL」が必要になるのは、例えば Elastic Beanstalk 上で実行するために構築したウェブアプリケーションのデモを行う場合などです。「Launch Now URL」により、パラメータを使用して、必要な情報を事前に Create Application ウィザードに追加することができます。この情報をウィザードに追加すると、誰でも URL リンクを使用して、短い手順でウェブアプリケーションソースで Elastic Beanstalk 環境を起動できます。つまり、ユーザーはアプリケーションソースバンドルの場所を手動でアップロードまたは指定する必要がありません。また、ウィザードに追加の情報を入力する必要もありません。

「Launch Now URL」では、アプリケーションの作成に必要な最低限の情報 (アプリケーション名、ソリューションスタック、インスタンスタイプ、および環境タイプ) が Elastic Beanstalk に渡されます。カスタムの「Launch Now URL」で明示的に指定されていない設定事項については、Elastic Beanstalk によりデフォルト値が使用されます。

「Launch Now URL」は、標準の URL 構文を使用します。詳細については、「RFC 3986 - Uniform Resource Identifier (URI): 一般的な構文」を参照してください。

URL パラメータ

URL には次のパラメータを含める必要があります。大文字と小文字は区別されます。

- region – AWS リージョンを指定します。Elastic Beanstalk でサポートされているリージョンのリストについては、「AWS 全般のリファレンス」の「AWS Elastic Beanstalk エンドポイントとクォータ」を参照してください。
- applicationName – アプリケーションの名前を指定します。Elastic Beanstalk コンソールでは、他のアプリケーションと区別できるように Elastic Beanstalk によってこのアプリケーション名が表示されます。デフォルトでは、アプリケーションの名前は環境の名前と環境 URL を基に作成されます。

- platform – 環境で使用するプラットフォームのバージョンを指定します。次のいずれかのメソッドを使用し、選択したものを URL エンコードします。
  - バージョンなしでプラットフォーム ARN を指定します。Elastic Beanstalk により、対応するプラットフォームのメジャーバージョンの最新プラットフォームバージョンが選択されます。例えば、最新の Python 3.6 プラットフォームバージョンを選択するには、`Python 3.6 running on 64bit Amazon Linux` を指定します。
  - プラットフォーム名を指定します。Elastic Beanstalk により、プラットフォームの最新言語ランタイムの最新バージョンが選択されます (例: `Python`)。

使用可能なすべてのプラットフォームとそのバージョンの説明については、「Elastic Beanstalk でサポートされているプラットフォーム」を参照してください。

AWS Command Line Interface (AWS CLI) を使用すると、すべての使用可能なプラットフォームのバージョンのリストとそれぞれの ARN を取得できます。`list-platform-versions` コマンドは、すべての使用可能なプラットフォームのバージョンに関する詳細情報を一覧表示します。`--filters` 引数を使用してリストの範囲を制限します。例えば、特定の言語のプラットフォームのバージョンのみを一覧表示するように制限できます。

次の例では、すべての Python プラットフォームバージョンのクエリを実行し、出力を一連のコマンドのパイプに送り込みます。結果は、プラットフォームのバージョン ARN のリスト (末尾に `/version` がない) となります。人間が読み取れる形式で、URL エンコードはありません。

```
$ aws elasticbeanstalk list-platform-versions --filters
 'Type="PlatformName",Operator="contains",Values="Python"' | grep PlatformArn | awk -
F '"' '{print $4}' | awk -F '/' '{print $2}'
Preconfigured Docker - Python 3.4 running on 64bit Debian
Preconfigured Docker - Python 3.4 running on 64bit Debian
Python 2.6 running on 32bit Amazon Linux
Python 2.6 running on 32bit Amazon Linux 2014.03
...
Python 3.6 running on 64bit Amazon Linux
```

次の例では、最後の例に Perl コマンドを追加し、出力を URL エンコードします。

```
$ aws elasticbeanstalk list-platform-versions --filters
 'Type="PlatformName",Operator="contains",Values="Python"' | grep PlatformArn | awk
 -F '"' '{print $4}' | awk -F '/' '{print $2}' | perl -MURI::Escape -ne 'chomp;print
 uri_escape($_),"\n"'
Preconfigured%20Docker%20-%20Python%203.4%20running%20on%2064bit%20Debian
```

```
Preconfigured%20Docker%20-%20Python%203.4%20running%20on%2064bit%20Debian
Python%202.6%20running%20on%2032bit%20Amazon%20Linux
Python%202.6%20running%20on%2032bit%20Amazon%20Linux%202014.03
...
Python%203.6%20running%20on%2064bit%20Amazon%20Linux
```

「Launch Now URL」は、オプションで次のパラメータを含めることができます。オプションのパラメータを「Launch Now URL」に含めなかった場合、Elastic Beanstalk ではデフォルトの値を使用してアプリケーションの作成および実行が行われます。sourceBundleUrl パラメータが含まれていない場合、Elastic Beanstalk は指定の platform に対応するデフォルトのサンプルアプリケーションを使用します。

- sourceBundleUrl – ウェブアプリケーションソースバンドルの場所を URL 形式で指定します。たとえば、Amazon S3 バケットにソースバンドルをアップロードした場合は、sourceBundleUrlパラメータの値を `https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject` のように指定できます。

> **Note**
>
> [sourceBundleUrl] パラメータの値を HTTP URL として指定できますが、ユーザーのウェブブラウザは、必要に応じて HTML URL エンコードのポリシーを使用して文字を変換します。

- environmentType] – 環境のタイプとして、負荷分散されたスケーラブルな環境であるか、単一のインスタンスであるかを指定します。詳細については、「」を参照してください環境タイプ パラメータ値として `LoadBalancing` または `SingleInstance` を指定できます。
- tierName - ウェブリクエストを処理するウェブアプリケーションか、バックグラウンドジョブを実行するウェブアプリケーションのどちらを環境でサポートしているかを指定します。詳細については、「」を参照してくださいElastic Beanstalk ワーカー環境 `WebServer` または `Worker` のどちらかを指定できます。
- instanceType - アプリケーションに最適な特性 (メモリサイズや CPU 能力など) を持つサーバーを選択します。Amazon EC2 インスタンスファミリーとタイプの詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスタイプ」を参照してください。各リージョンで利用可能なインスタンスタイプの詳細については、「Amazon EC2 ユーザーガイド」の「利用可能なインスタンスタイプ」を参照してください。

- withVpc - Amazon VPC で環境を作成するかどうかを指定します。`true` または `false` のどちらかを指定できます。Amazon VPC で Elastic Beanstalk を使用する方法については、「Amazon VPC で Elastic Beanstalk を使用する」を参照してください。
- withRds - この環境で Amazon RDS データベースインスタンスを作成するかどうかを指定します。詳細については、「」を参照してくださいAmazon RDS で Elastic Beanstalk を使用する `true` または `false` のどちらかを指定できます。
- rdsDBEngine - この環境の Amazon EC2 インスタンスで使用するデータベースエンジンを指定します。`mysql`、`oracle-sel`、`sqlserver-ex`、`sqlserver-web`、または `sqlserver-se` を指定できます。デフォルト値は `mysql` です。
- rdsDBAllocatedStorage – 割り当て済みデータベースストレージのサイズをギガバイト (GB) 単位で指定します。次の値を指定できます。
  - MySQL – `5` ~ `1024`。デフォルト: `5`。
  - Oracle – `10` ~ `1024`。デフォルト: `10`。
  - Microsoft SQL Server Express Edition – `30`。
  - Microsoft SQL Server Web Edition – `30`。
  - Microsoft SQL Server Standard Edition – `200`。
- rdsDBInstanceClass - データベースインスタンスタイプを指定します。デフォルト値は `db.t2.micro` (Amazon VPC で実行されていない環境では `db.m1.large`) です。Amazon RDS でサポートされているデータベースインスタンスクラスのリストについては、Amazon Relational Database Service ユーザーガイドの「DB インスタンスクラス」を参照してください。
- rdsMultiAZDatabase - Elastic Beanstalk で複数のアベイラビリティーゾーンを対象としたデータベースインスタンスを作成する必要があるかどうかを指定します。`true` または `false` のどちらかを指定できます。Amazon RDS を使用した複数のアベイラビリティーゾーンのデプロイについては、Amazon Relational Database Service ユーザーガイドの「リージョンとアベイラビリティーゾーン」を参照してください。
- rdsDBDeletionPolicy - 環境の終了時に、データベースインスタンスを削除するか、そのスナップショットを取得するかを指定します。`Delete` または `Snapshot` のどちらかを指定できます。

例

「Launch Now URL」の例を次に示します。アプリケーションを構築した後、ユーザーがそれを使用することができます。例えば、URL をウェブページまたはトレーニング教材に組み込むことができます。ユーザーが「Launch Now URL」を使用してアプリケーションを作成する場合、Elastic Beanstalk のアプリケーションの作成ウィザードで追加の入力は必要ありません。

```
https://console.aws.amazon.com/elasticbeanstalk/home?region=us-west-2#/newApplication?
applicationName=YourCompanySampleApp
      &platform=PHP%207.3%20running%20on%2064bit%20Amazon%20Linux&sourceBundleUrl=
        http://s3.amazonaws.com/amzn-s3-demo-bucket/
myobject&environmentType=SingleInstance&tierName=WebServer
      &instanceType=m1.small&withVpc=true&withRds=true&rdsDBEngine=

 postgres&rdsDBAllocatedStorage=6&rdsDBInstanceClass=db.m1.small&rdsMultiAZDatabase=
          true&rdsDBDeletionPolicy=Snapshot
```

Launch Now URL を使用するには

1. [Launch Now URL] を選択します
2. Elastic Beanstalk コンソールが開いたら [ウェブアプリケーションの作成] ページで [確認と起動] を選択します。アプリケーションを作成し、アプリケーションの実行環境を起動するために Elastic Beanstalk が使用する設定が表示されます。
3. [設定] ページで、[アプリの作成] を選択してアプリケーションを作成します。

Elastic Beanstalk 環境のグループを作成および更新する

AWS Elastic Beanstalk ComposeEnvironments API を使用すると、単一のアプリケーション内で Elastic Beanstalk 環境のグループを作成および更新できます。グループ内の各環境は、サービス対応アーキテクチャアプリケーションごとにそれぞれのコンポーネントを実行できます。`Compose Environments` API は、アプリケーションバージョンおよびオプションでグループ名のリストを取得します。Elastic Beanstalk は、アプリケーションバージョンごとに環境を作成します。またはこの環境がすでに存在する場合には、アプリケーションバージョンを環境にデプロイします。

Elastic Beanstalk の環境間にリンクを作成すると、1 つの環境を他の環境の依存関係として指定できます。`Compose Environments` API で環境のグループを作成する場合、依存する側の環境が Elastic Beanstalk によって作成されるのは、依存関係既に有効になっている場合に限られます。環境リンクの詳細については、「Elastic Beanstalk 環境間のリンクの作成」を参照してください。

`Compose Environments` API は、環境マニフェストを使用して環境グループで共有される詳細設定を保存します。それぞれの構成アプリケーションには、アプリケーションのソースバンドル内に `env.yaml` 形式の設定ファイルが存在することが必須であり、これによってその環境を作成するために使用されるパラメータを特定します。

Compose Environments は、各構成アプリケーションの環境マニフェストで EnvironmentName と SolutionStack が特定されていることを要件とします。

Compose Environments API は、Elastic Beanstalk コマンドラインインターフェイス (EB CLI)、AWS CLI、または SDK で使用できます。EB CLI の説明については、「EB CLI で複数の Elastic Beanstalk 環境をグループとして管理する」を参照してください。

Compose Environments API の使用

たとえば、画像やビデオをユーザーが Amazon Simple Storage Service (Amazon S3) にアップロードして管理できる、Media Library という名前のアプリケーションを作成することができます。このアプリケーションにはフロントエンド環境、front、があり、ユーザーがそれぞれのファイルをアップロード・ダウンロード、ライブラリの参照、そしてバッチ処理ジョブを開始することができるウェブアプリケーションを実行します。

ジョブを直接処理する代わりに、フロントエンドアプリケーションは Amazon SQS キューにジョブを追加します。2 番目の環境、worker、はキューからジョブを取り出して処理します。 worker は高パフォーマンス GPU の G2 インスタンスタイプを使用しますが、一方 front はさらにコストパフォーマンスに優れた一般インスタンスタイプで実行できます。

プロジェクトフォルダ、Media Library、をそれぞれのコンポーネントごとのディレクトリに分けて、各ディレクトリにはそれぞれのソースコードを含む環境定義ファイル (env.yaml) があるように設定できます。

```
~/workspace/media-library
|-- front
|   `-- env.yaml
`-- worker
    `-- env.yaml
```

次のリストは、各コンポーネントアプリケーションの env.yaml ファイルを示します。

~/workspace/media-library/front/env.yaml

```
EnvironmentName: front+
EnvironmentLinks:
  "WORKERQUEUE" : "worker+"
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentTier:
  Name: WebServer
```

```
  Type: Standard
SolutionStack: 64bit Amazon Linux 2015.09 v2.0.4 running Java 8
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: m4.large
```

~/workspace/media-library/worker/env.yaml

```
EnvironmentName: worker+
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentTier:
  Name: Worker
  Type: SQS/HTTP
SolutionStack: 64bit Amazon Linux 2015.09 v2.0.4 running Java 8
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: g2.2xlarge
```

フロントエンド (`front-v1`) とワーカー (`worker-v1`) アプリケーションコンポーネント用のアプリケーションバージョンを作成したら、そのバージョン名の `Compose Environments` API を呼び出すことができます。この例では、 AWS CLI を使用して API を呼び出します。

```
# Create application versions for each component:
~$ aws elasticbeanstalk create-application-version --application-name media-
library --version-label front-v1 --process --source-bundle S3Bucket="amzn-s3-demo-
bucket",S3Key="front-v1.zip"
  {
    "ApplicationVersion": {
        "ApplicationName": "media-library",
        "VersionLabel": "front-v1",
        "Description": "",
        "DateCreated": "2015-11-03T23:01:25.412Z",
        "DateUpdated": "2015-11-03T23:01:25.412Z",
        "SourceBundle": {
            "S3Bucket": "amzn-s3-demo-bucket",
            "S3Key": "front-v1.zip"
        }
    }
  }
~$ aws elasticbeanstalk create-application-version --application-name media-library
 --version-label worker-v1 --process --source-bundle S3Bucket="amzn-s3-demo-
bucket",S3Key="worker-v1.zip"
```

```
  {
    "ApplicationVersion": {
        "ApplicationName": "media-library",
        "VersionLabel": "worker-v1",
        "Description": "",
        "DateCreated": "2015-11-03T23:01:48.151Z",
        "DateUpdated": "2015-11-03T23:01:48.151Z",
        "SourceBundle": {
            "S3Bucket": "amzn-s3-demo-bucket",
            "S3Key": "worker-v1.zip"
        }
    }
  }
# Create environments:
~$ aws elasticbeanstalk compose-environments --application-name media-library --group-
name dev --version-labels front-v1 worker-v1
```

3 番目の呼び出しでは、`front-dev` と `worker-dev` の 2 つの環境を作成します。この API は、`EnvironmentName` ファイルに特定された `env.yaml` と `group name` 呼び出しに特定された `Compose Environments` オプションをハイフンで区切って連結した環境の名前を作成します。ハイフンを含めたこの 2 つのオプションの合計は、環境の名前に使用できる最大限の 23 文字を超えることはできません。

`front-dev` 環境で実行されるアプリケーションは `worker-dev` 変数を読み取ることにより、`WORKERQUEUE` 環境にアタッチされた Amazon SQS キューの名前にアクセスできます。環境リンクの詳細については、「Elastic Beanstalk 環境間のリンクの作成」を参照してください。

EB CLI で複数の Elastic Beanstalk 環境をグループとして管理する

EB CLI を使用して AWS Elastic Beanstalk 環境のグループを作成し、それぞれがサービス指向アーキテクチャアプリケーションの個別のコンポーネントを実行します。EB CLI は、ComposeEnvironments API を使用してこのようなグループを管理します。

> **Note**
>
> 環境グループは、マルチコンテナの Docker 環境の複数のコンテナとは異なります。環境グループでは、アプリケーションの各コンポーネントは、別の Elastic Beanstalk 環境で独自の専用 Amazon EC2 インスタンスのセットと共に実行されます。各コンポーネントは個別にスケールできます。マルチコンテナの Docker では、単一の環境にアプリケーションの複数のコンポーネントを結合します。すべてのコンポーネントは同じ Amazon EC2 インスタンスの

セットを共有し、各インスタンスは複数の Docker コンテナを実行します。アプリケーションのニーズに基づいて、これらのいずれかのアーキテクチャを選択します。
マルチコンテナの Docker の詳細については、「Elastic Beanstalk で ECS マネージド Docker プラットフォームブランチを使用する」を参照してください。

次のフォルダ構造にアプリケーションコンポーネントを整理します。

```
~/project-name
|-- component-a
|   `-- env.yaml
`-- component-b
    `-- env.yaml
```

それぞれのサブフォルダには、独自の環境と `env.yaml` という環境定義ファイルを実行する 1 つのアプリケーションの個別のコンポーネントのソースコードが含まれます。`env.yaml` 形式の詳細については、「マニフェスト環境 (`env.yaml`)」を参照してください。

`Compose Environments` API を使用するためには、まずプロジェクトフォルダから eb init を実行し、`--modules` オプションでコンポーネントがあるフォルダの名前をそれぞれのコンポーネントごとに指定します。

```
~/workspace/project-name$ eb init --modules component-a component-b
```

EB CLI は、各コンポーネントを設定するプロンプトを表示し、続いて各コンポーネントフォルダに `.elasticbeanstalk` ディレクトリを作成します。EB CLI は親ディレクトリに設定ファイルを作成しません。

```
~/project-name
|-- component-a
|   |-- .elasticbeanstalk
|   `-- env.yaml
`-- component-b
    |-- .elasticbeanstalk
    `-- env.yaml
```

続いて、コンポーネントごとに、作成する環境のリストがある eb create コマンドを実行します:

```
~/workspace/project-name$ eb create --modules component-a component-b --env-group-suffix group-name
```

このコマンドは、各コンポーネントの環境を作成します。環境の名前は、EnvironmentName ファイルに特定された env.yaml とグループ名をハイフンで区切って連結して作成されます。ハイフンを含めたこの 2 つのオプションの合計は、環境の名前に使用できる最大限の 23 文字を超えることはできません。

環境を更新するためには、eb deploy コマンドを使用します。

```
~/workspace/project-name$ eb deploy --modules component-a component-b
```

コンポーネントごとを個別に、あるいはグループとして更新できます。--modules オプションを使用して更新するコンポーネントを指定します。

EB CLI は、eb create で使用したグループ名を、branch-defaults の下にある EB CLI 設定ファイルの /.elasticbeanstalk/config.yml セクションに保存します。アプリケーションを別々のグループにデプロイする場合は、--env-group-suffix 実行時に eb deploy オプションを使用します。グループが既に存在しない場合には、EB CLI は環境の新しいグループを作成します。

```
~/workspace/project-name$ eb deploy --modules component-a component-b --env-group-suffix group-2-name
```

環境を終了するには、各モジュールのフォルダ内で eb terminate を実行します。デフォルトでは、実行されているその他の環境と依存関係にある環境を終了しようとする際に、EB CLI によって エラーが表示されます。まず依存した環境を終了するか、あるいは --ignore-links オプションでデフォルトの動作を変更します。

```
~/workspace/project-name/component-b$ eb terminate --ignore-links
```

Elastic Beanstalk 環境へのアプリケーションのデプロイ

AWS Elastic Beanstalk コンソールを使用して、更新されたソースバンドルをアップロードして Elastic Beanstalk 環境にデプロイするか、以前にアップロードしたバージョンを再デプロイできます。

各デプロイはデプロイ ID で識別されます。デプロイ ID は 1 から始まり、デプロイするか、インスタンスの設定を変更するたびに、1 ずつ増えます。Elastic Beanstalk では、拡張ヘルスレポートを有

効にした場合、インスタンスのヘルスステータスのレポート時に、ヘルスコンソールと EB CLI の両方でデプロイ ID が表示されます。デプロイ ID は、ローリング更新が失敗したときにお客様の環境の状態を調べるために役立ちます。

Elastic Beanstalk には、いくつかのデプロイポリシーと設定が用意されています。ポリシーの設定と追加設定の詳細については、「the section called “デプロイオプション”」を参照してください。以下の表では、ポリシーとそのポリシーをサポートする環境の種類を示しています。

サポートされているデプロイメントポリシー

デプロイメントポリシー	負荷分散された環境	単一インスタンス環境	レガシー Windows サーバー環境†
All at once	✓ はい	✓ はい	✓ はい
ローリング	✓ はい	×いいえ	✓ はい
Rolling with an additional batch (追加バッチによるローリング)	✓ はい	×いいえ	×いいえ
イミュータブル	✓ はい	✓ はい	×いいえ
トラフィック分割	✓ はい (Application Load Balancer)	×いいえ	×いいえ

† この表では、レガシー Windows Server 環境は、IIS 8.5 より前の IIS バージョンを使用する Windows Server プラットフォーム設定に基づいています。

> **Warning**
>
> 一部のポリシーでは、デプロイ時または更新時にすべてのインスタンスが置き換えられます。これにより、累積したすべての Amazon EC2 バーストバランスが失われます。次の場合に発生します。
>
> - インスタンスの置換を有効にしたマネージドプラットフォームの更新
> - イミュータブルな更新

- イミュータブルな更新またはトラフィック分割を有効にしたデプロイ

デプロイポリシーの選択

アプリケーションに適切なデプロイポリシーを選択することは、いくつかの考慮事項のトレードオフであり、特定のニーズに左右されます。「the section called “デプロイオプション”」ページには、各ポリシーについての詳細情報があり、一部のポリシーについては仕組みが詳しく説明されています。

以下のリストには、さまざまなデプロイポリシーについての概要情報があり、関連する考慮事項が追加されています。

- All at once (一度にすべて) - 最も迅速なデプロイ方法。サービスの短期間の停止が許容され、迅速なデプロイが重要である場合に適しています。この方法では、Elastic Beanstalk は新しいアプリケーションバージョンを各インスタンスにデプロイします。その後、ウェブプロキシまたはアプリケーションサーバーを再起動する必要があります。したがって、ユーザーがアプリケーションを短時間使用できなくなる (可用性が低くなる) ことがあります。
- Rolling (ローリング) - ダウンタイムを回避し、可用性の低下を最小限に抑えます。ただし、デプロイ時間は長くなります。サービスの完全停止期間が許容されない場合に適しています。この方法では、アプリケーションはお客様の環境で一度に 1 バッチのインスタンスにデプロイされます。ほとんどの帯域幅はデプロイ全体で保持されます。
- Rolling with additional batch (追加バッチによるローリング) - 可用性の低下を回避します。ただし、[Rolling (ローリング)] 方法よりもデプロイ時間が長くなります。デプロイ全体で同じ帯域幅を保持する必要がある場合に適しています。この方法では、Elastic Beanstalk はインスタンスの追加バッチを起動し、ローリングデプロイを実行します。追加バッチの起動に時間をかけることで、デプロイ全体で同じ帯域幅が確実に保持されます。
- Immutable (イミュータブル) - 低速のデプロイ方法。既存のインスタンスを更新するのではなく、常に新しいアプリケーションバージョンを新しいインスタンスにデプロイします。また、デプロイが失敗した場合に迅速かつ安全にロールバックできるという追加の利点もあります。この方法では、Elastic Beanstalk はイミュータブルな更新を実行してアプリケーションをデプロイします。イミュータブルな更新では、環境内で 2 番目の Auto Scaling グループが起動し、新しいインスタンスがヘルスチェックに合格するまで、新しいバージョンが旧バージョンと並行してトラフィックを提供します。
- Traffic splitting (トラフィック分割) - Canary テストのデプロイ方法。受信トラフィックの一部を使用して新しいアプリケーションバージョンのヘルスをテストしながら、残りのトラフィックを古いアプリケーションバージョンで処理され続けるようにする場合に適しています。

以下の表は、デプロイ方法のプロパティを比較したものです。

デプロイ方法

方法	デプロイ失敗の影響	デプロイ所要時間	ゼロダウンタイム	DNS の変更なし	ロールバックプロセス	コードのデプロイ先
All at once	ダウンタイム		×いいえ	✓ はい	手動再デプロイ	既存のインスタンス
ローリング	サービス停止状態の単一のバッチ。新しいアプリケーションバージョンを実行している、失敗前のすべてのバッチ。		✓ はい	✓ はい	手動再デプロイ	既存のインスタンス
Rolling with an additional batch (追加バッチによるローリング)	最初のバッチが失敗した場合、影響は最小限。それ以外の場合は [ローリング] と類似。		✓ はい	✓ はい	手動再デプロイ	新規および既存のインスタンス
イミュー	最小限		✓ はい	✓ はい	新しいイ	新規のイ

方法	デプロイ失敗の影響	デプロイ所要時間	ゼロダウンタイム	DNSの変更なし	ロールバックプロセス	コードのデプロイ先
タブル					ンスタンスの終了	ンスタンス
トラフィック分割	新しいバージョンにルーティングされて一時的に影響を受けたクライアントトラフィックの割合		✓ はい	✓ はい	トラフィックの再ルーティングと新しいインスタンスの終了	新規のインスタンス
Blue/Green	最小限		✓ はい	×いいえ	URLのスワップ	新規のインスタンス

† バッチサイズにより異なります。

†† [evaluation time (評価時間)] オプションの設定によって異なります。

新しいアプリケーションバージョンのデプロイ

環境のダッシュボードからデプロイを実行できます。

新しいアプリケーションバージョンを Elastic Beanstalk 環境にデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [アップロードとデプロイ] を選択します。
4. 画面上のフォームを使用して、アプリケーションソースバンドルをアップロードします。
5. [デプロイ] を選択します。

以前のバージョンの再デプロイ

アプリケーションの以前にアップロードしたバージョンを、アプリケーションのバージョンのページからその環境のいずれかにデプロイできます。

既存のアプリケーションバージョンを既存の環境にデプロイするには

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. ナビゲーションペインで、アプリケーション名を見つけ、[アプリケーションバージョン] を選択します。
4. デプロイするアプリケーションバージョンを選択します。
5. [アクション]、[デプロイ] の順に選択します。
6. 環境を選択してから、[デプロイ] を選択します。

アプリケーションをデプロイするその他の方法

頻繁にデプロイする場合は、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用してお客様の環境を管理することを検討してください。EB CLI では、ソースコードと共にリポジトリを作成します。また、1 つのコマンドで、ソースバンドルを作成し、Elastic Beanstalk にアップロードして、デプロイすることもできます。

リソースの設定変更や、旧バージョンと同時に実行できない新しいバージョンに依存するデプロイの場合、新しいバージョンで新しい環境を起動し、ブルー/グリーンデプロイ 向けに CNAME スワップを実行します。

ビルド、テスト、デプロイのプロセスを自動化するために、Elastic Beanstalk 環境との継続的インテグレーションと継続的デプロイ (CI/CD) を実装できます。詳細については、「Elastic Beanstalk 環境との CI/CD 統合の実装」を参照してください。

デプロイポリシーと設定

AWS Elastic Beanstalk には、デプロイポリシー (一度にすべて、ローリング、追加のバッチによるローリング、イミュータブル、トラフィック分割) や、デプロイ中のバッチサイズとヘルスチェックの動作を設定できるオプションなど、デプロイの処理方法に関するいくつかのオプションが用意されています。デフォルトでは、環境は一括デプロイを使用します。EB CLI で環境を作成し、それがスケーラブルな環境である (`--single` オプションを指定していない) 場合は、ローリングデプロイを使用します。

rolling deployments (ローリングデプロイ) では、Elastic Beanstalk はお客様の環境の Amazon EC2 インスタンスをバッチに分割し、アプリケーションの新しいバージョンを一度に 1 バッチのインスタンスにデプロイします。残りのインスタンスは、お客様の環境でアプリケーションの古いバージョンを実行し続けます。つまりローリングデプロイ中は、アプリケーションの古いバージョンでリクエストを処理するインスタンスもあり、新しいバージョンでリクエストを処理するインスタンスも存在します。詳細については、「the section called “ローリングデプロイの仕組み”」を参照してください。

デプロイ中に総容量を維持するには、インスタンスがサービス停止になる前に、インスタンスの新しいバッチを起動するよう環境を設定できます。このオプションは、rolling deployment with an additional batch (追加バッチによるローリングデプロイ) と呼ばれます。デプロイが完了すると、Elastic Beanstalkがインスタンスの追加バッチを終了します。

イミュータブルなデプロイは、イミュータブルな更新を実行して、古いバージョンを起動しているインスタンスと並行して、別の Auto Scaling グループにあるアプリケーションの新しいバージョンを起動している新しいインスタンスのフルセットを起動します。[Immutable] デプロイは、部分的に完了したローリングデプロイにより発生する問題を防止できます。新しいインスタンスがヘルスチェックに合格しなかった場合、Elastic Beanstalk はそれを終了し、元のインスタンスをそのまま残します。

[Traffic-splitting deployments (トラフィック分割デプロイ)] では、アプリケーションのデプロイの一部として Canary テストを実施できます。Elastic Beanstalk は、トラフィック分割デプロイ時に、イ

ミュータブルなデプロイ時と同様に、新しいインスタンスのフルセットを起動します。次に、指定された評価期間にわたり、指定された割合の受信クライアントトラフィックを新しいアプリケーションバージョンに転送します。新しいインスタンスが正常である限り、Elastic Beanstalk はすべてのトラフィックを新しいインスタンスに転送し、古いインスタンスを終了します。新しいインスタンスがヘルスチェックに合格しなかったり、デプロイの中止が選択されたりすると、Elastic Beanstalk はトラフィックを古いインスタンスに戻し、新しいインスタンスを終了します。サービスが中断されることはありません。詳細については、「the section called “トラフィック分割デプロイの仕組み”」を参照してください。

> **Warning**
>
> 一部のポリシーでは、デプロイ時または更新時にすべてのインスタンスが置き換えられます。これにより、累積したすべての Amazon EC2 バーストバランスが失われます。次の場合に発生します。
>
> - インスタンスの置換を有効にしたマネージドプラットフォームの更新
> - イミュータブルな更新
> - イミュータブルな更新またはトラフィック分割を有効にしたデプロイ

アプリケーションがすべてのヘルスチェックに合格しないものの、低いヘルスステータスで正しく起動する場合、インスタンスが合格できるヘルスチェックのステータスを、`Warning` のような低いものに [Healthy threshold] オプションを使って変更し、合格させることができます。ヘルスチェックをパスできずにデプロイに失敗し、ヘルスステータスに関係なく強制的に更新する必要がある場合は、[Ignore health check] オプションを選択してヘルスチェックを無視します。

ローリング更新のバッチサイズを指定すると、Elastic Beanstalk もまたローリングアプリケーション再起動の際にその値を使用します。ダウンタイムなしに、環境のインスタンスで実行しているプロキシおよびアプリケーション サーバーの再起動が必要な場合は、ローリング再起動を使用します。

アプリケーションデプロイの設定

環境マネジメントコンソールで、バッチ処理されるアプリケーションバージョンのデプロイの有効化や設定を行うには、環境の [Configuration] ページにある [Updates and Deployments] を編集します。

デプロイ (コンソール) を設定する

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。

3. ナビゲーションペインで、[設定] を選択します。
4. [Rolling updates and deployments (ローリング更新とデプロイ)] 設定カテゴリで [Edit (編集)] を選択します。
5. [アプリケーションのデプロイ] セクションで、[デプロイメントポリシー]、バッチ設定、ヘルスチェックオプションを選択します。
6. ページの最下部で [適用] を選択し変更を保存します。

[Rolling updates and deployments] (ローリング更新とデプロイ) ページの [Application deployments] (アプリケーションのデプロイ) セクションには、アプリケーションのデプロイに関する次のオプションがあります。

- [Deployment policy (デプロイポリシー)] - 以下のデプロイオプションから選択します。
  - [All at once (一度に)] - 同時にすべてのインスタンスに新しいバージョンをデプロイします。環境内のすべてのインスタンスは、デプロイが実行される間、短時間ですがサービス停止状態になります。
  - [Rolling (ローリング)] - バッチに新しいバージョンをデプロイします。デプロイフェーズ中、バッチはサービス停止状態になり、バッチのインスタンスによる環境容量への負荷を低減します。
  - [Rolling with additional batch (追加バッチによるローリング)] - バッチに新しいバージョンをデプロイしますが、デプロイ中に総容量を維持するため、インスタンスの新しいバッチをまず起動します。
  - [Immutable (イミュータブル)] - イミュータブルな更新を実行し、新しいバージョンをインスタンスの新しいグループにデプロイします。
  - [Traffic splitting (トラフィック分割)] - アプリケーションの新しいバージョンをインスタンスの新しいグループにデプロイし、受信クライアントトラフィックをアプリケーションの既存のバージョンと新しいバージョンとの間で一時的に分割します。

[Rolling (ローリング)] および [Rolling with additional batch (追加バッチによるローリング)] デプロイポリシーでは、以下を設定できます。

- [Batch size (バッチサイズ)] - 各バッチでデプロイする一連のインスタンスのサイズ。

  Auto Scaling グループ (最大 100%) で EC2 インスタンスの総数の割合を構成するには、[Percentage (パーセント)] を選択するか、[Fixed (固定)] を選択して固定数のインスタンスを設定します (環境の Auto Scaling 設定の最大インスタンス数まで)。

[Traffic splitting (トラフィック分割)] デプロイポリシーでは、以下を設定できます。

- [Traffic split (トラフィック分割)] - デプロイする新しいアプリケーションバージョンを実行している環境インスタンスに Elastic Beanstalk がシフトする受信クライアントトラフィックの初期の割合。
- [Traffic splitting evaluation time (トラフィック分割評価時間)] - 初回のデプロイが正常に完了してから、デプロイする新しいアプリケーションバージョンにすべての受信クライアントトラフィックをシフトするまで、Elastic Beanstalk が待機する時間 (分単位)。

[Deployment preferences] セクションには、ヘルスチェック関連のオプションが含まれています。

- [Ignore health check (ヘルスチェックの無視)] - バッチが [Command timeout (コマンドタイムアウト)] の時間内に正常な状態にならなかった場合に、デプロイがロールバックするのを防ぎます。
- [Healthy threshold (ヘルシーしきい値)] - ローリングデプロイやローリング更新、イミュータブルな更新中に、インスタンスが正常と見なされるしきい値を引き下げます。
- [Command timeout (コマンドタイムアウト)] - デプロイがキャンセルされるまで、または [Ignore health check (ヘルスチェックの無視)] が設定されている場合は次のバッチの処理に移るまで、インスタンスが正常な状態になるために待機する秒数。

ローリングデプロイの仕組み

バッチを処理する際、Elastic Beanstalk はバッチ内のすべてのインスタンスをロードバランサーからデタッチし、新しいアプリケーションバージョンをデプロイしてから、再アタッチします。Connection Draining が有効になっていると、Elastic Beanstalk は、デプロイを開始する前に、Amazon EC2 インスタンスから既存の接続をドレインします。

バッチ内のインスタンスがロードバランサーに再アタッチされると、Elastic Load Balancing は、これらのインスタンスが最小限の数の Elastic Load Balancing ヘルスチェック ([Healthy check count threshold (ヘルスチェック数のしきい値)] の値) に合格するのを待ってから、これらのインスタンスへのトラフィックのルーティングを開始します。ヘルスチェック URL が設定されていなければ、これはすぐに発生する可能性があります。インスタンスが TCP 接続を受け付けるとすぐにヘルスチェックに合格するからです。ヘルスチェック URL が設定されていると、`200 OK` ステータスコードがヘルスチェック URL への `HTTP GET` リクエストの応答として返されるまで、ロードバランサーはトラフィックを更新されたインスタンスにルーティングしません。

Elastic Beanstalk は、バッチ内のすべてのインスタンスが正常な状態になるまで待ち、その後に次のバッチを処理します。基本的なヘルスレポートでは、インスタンスのヘルスは Elastic Load Balancing ヘルスチェックのステータスによって決まります。バッチ内のすべてのインスタンスが、Elastic Load Balancing によって正常な状態であると見なされるために十分な数のヘルスチェックに合格すると、バッチは正常であると認められます。拡張ヘルスレポートが有効になっていると、Elastic Beanstalk はいくつかの他の要因 (着信リクエストの結果など) も考慮します。ウェブサーバー環境の拡張ヘルスレポートでは、すべてのインスタンスが 2 分間にわたって連続して行われる 12 のヘルスチェック (ワーカー環境の場合は 3 分間にわたって 18 のヘルスチェック) に OK ステータスで合格する必要があります。

インスタンスのバッチがコマンドタイムアウト内に正常にならない場合、デプロイは失敗します。デプロイの失敗後、失敗の原因については、お客様の環境内のインスタンスのヘルスを確認してください。次に、アプリケーションの固定または既知の適切なバージョンを使用して別のデプロイメントを実行して、ロールバックします。

1 つ以上のバッチが正常に完了した後にデプロイが失敗した場合、完了したバッチはアプリケーションの新しいバージョンを実行する一方で、保留中のバッチは古いバージョンを実行し続けます。コンソールの [Health] ページで、お客様の環境内のインスタンスで実行されているバージョンを特定できます。このページには、お客様の環境内の各インスタンスで実行されている最新のデプロイのデプロイ ID が表示されます。失敗したデプロイのインスタンスを終了した場合、Elastic Beanstalk ではそれらのインスタンスが、成功した最新のデプロイのアプリケーションバージョンを実行中のインスタンスに置き換えられます。

トラフィック分割デプロイの仕組み

トラフィック分割デプロイでは、Canary テストを実施できます。一部の受信クライアントトラフィックを新しいアプリケーションバージョンに転送して、そのアプリケーションの状態を確認してから、その新しいバージョンにコミットし、すべてのトラフィックを転送します。

トラフィック分割デプロイ中に、Elastic Beanstalk は別の一時的 Auto Scaling グループに新しい一連のインスタンスを作成します。次に、Elastic Beanstalk は、環境の受信トラフィックの特定の割合を新しいインスタンスに転送するようにロードバランサーに指示します。その後、設定された期間、Elastic Beanstalk は新しい一連のインスタンスのヘルスを追跡します。ここまで問題がなければ、Elastic Beanstalk は残りのトラフィックを新しいインスタンスにシフトし、それらのインスタンスを環境の元の Auto Scaling グループにアタッチして、古いインスタンスを置き換えます。次に、Elastic Beanstalk は古いインスタンスをクリーンアップ (終了) して、一時的 Auto Scaling グループを削除します。

> **Note**
>
> トラフィック分割デプロイ中は、環境の容量は変わりません。Elastic Beanstalk は、デプロイの開始時に元の Auto Scaling グループにあるのと同じ数のインスタンスを一時的 Auto Scaling グループで起動します。その後、デプロイ期間中、両方の Auto Scaling グループのインスタンスが一定数に保たれます。環境のトラフィック分割評価時間を設定するときは、以下のことを考慮してください。

以前のアプリケーションバージョンへのデプロイのロールバックは迅速であり、クライアントトラフィックへのサービスには影響しません。新しいインスタンスがヘルスチェックに合格しなかったり、デプロイの中止が選択されたりすると、Elastic Beanstalk はトラフィックを古いインスタンスに戻し、新しいインスタンスを終了します。Elastic Beanstalk コンソールの環境概要ページを使用し、[Environment actions (環境のアクション)] で [Abort current operation (現在のオペレーションを中止)] を選択することで、デプロイを中止できます。AbortEnvironmentUpdate API または同等の AWS CLI コマンドを呼び出すこともできます。

トラフィック分割デプロイには、Application Load Balancer が必要です。Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、Elastic Beanstalk はデフォルトでこのロードバランサータイプを使用します。

デプロイオプションの名前空間

`aws:elasticbeanstalk:command` 名前空間にある設定オプションを使用して、デプロイを設定できます。トラフィック分割ポリシーを選択した場合、このポリシーの追加のオプションが `aws:elasticbeanstalk:trafficsplitting` 名前空間で使用できます。

`DeploymentPolicy` オプションを使用して、デプロイの種類を設定します。サポートされる値は次のとおりです。

- `AllAtOnce` - ローリングデプロイを無効にし、常に同時にすべてのインスタンスにデプロイを実行します。
- `Rolling` - 標準的なローリングデプロイを有効にします。
- `RollingWithAdditionalBatch` - 総容量を維持するために、デプロイ開始前にインスタンスの追加バッチを起動します。
- `Immutable` - すべてのデプロイ時にイミュータブルな更新を実行します。

- `TrafficSplitting` - トラフィック分割デプロイを実行して、アプリケーションのデプロイの Canary テストを実施します。

ローリングデプロイを有効にした場合は、`BatchSize` オプションと `BatchSizeType` オプションも設定して各バッチサイズを設定します。たとえば、各バッチのすべてのインスタンスの 25 パーセントをデプロイするには、次のオプションと値を指定します。

Example .ebextensions/rolling-updates.config

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: Rolling
    BatchSizeType: Percentage
    BatchSize: 25
```

実行中のインスタンスの数に関係なく、バッチごとに 5 つのインスタンスにデプロイを実行し、サービス停止状態のすべてのインスタンスをプルする前に、新しいバージョンを実行している 5 つのインスタンスの追加バッチを起動するには、次のオプションの値を指定します。

Example .ebextensions/rolling-additionalbatch.config

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: RollingWithAdditionalBatch
    BatchSizeType: Fixed
    BatchSize: 5
```

ヘルスチェックしきい値が [Warning] の各デプロイにイミュータブルな更新を実行し、バッチのインスタンスが 15 分のタイムアウト時間内にヘルスチェックにパスできなかったとしても、デプロイを実行するには、次のオプションと値を指定します。

Example .ebextensions/immutable-ignorehealth.config

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: Immutable
    HealthCheckSuccessThreshold: Warning
    IgnoreHealthCheck: true
    Timeout: "900"
```

トラフィック分割デプロイを実行し、クライアントトラフィックの 15% を新しいアプリケーションバージョンに転送して、10 分間にわたってヘルスを評価するには、以下のオプションと値を指定します。

Example .ebextensions/traffic-splitting.config

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: TrafficSplitting
  aws:elasticbeanstalk:trafficsplitting:
    NewVersionPercent: "15"
    EvaluationTime: "10"
```

EB CLI および Elastic Beanstalk コンソールでは、上記のオプションに推奨値が適用されます。設定ファイルを使用して同じファイルを設定する場合は、これらの設定を削除する必要があります。詳細については、「推奨値」を参照してください。

Elastic Beanstalk を使用したブルー/グリーンデプロイ

はアプリケーションバージョンを更新するときにインプレース更新 AWS Elastic Beanstalk を実行するため、アプリケーションが短時間ユーザーにアクセスできなくなる可能性があります。これを避けるためには、Blue-Green Deployment を実行します。これを行うためには、個別の環境に新しいバージョンをデプロイしてから、2 つの環境の CNAME を入れ替えて、直ちに新しいバージョンにトラフィックをリダイレクトします。

環境を互換性のないプラットフォームバージョンに更新する場合は、Blue-Green Deployment も必要です。詳細については、「the section called “プラットフォームの更新”」を参照してください。

Blue-Green Deployment では、アプリケーションがプロダクションデータベースを使用する場合、お客様の環境がそのデータベースから独立して実行される必要があります。Elastic Beanstalk がユーザーに代わって作成したデータベースが環境に含まれている場合、特定のアクションを実行しない限り、環境のデータベースと接続は保持されません。保持するデータベースがある場合は、Elastic Beanstalk データベースのライフサイクルオプションのいずれかを使用します。Retain オプションを選択すると、データベースをデカップリングした後でデータベースと環境を引き続き動作させることができます。詳細については、このガイドの環境の設定の章の「データベースのライフサイクル」を参照してください。

Elastic Beanstalk で管理されていない Amazon RDS インスタンスに接続するようにアプリケーションを設定する方法については、「Amazon RDS で Elastic Beanstalk を使用する」を参照してください。

Blue-Green Deployment を実行するには

1. Elastic Beanstalk 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. 現在の環境のクローンを作成するか、新しい環境を起動して希望のプラットフォームのバージョンを実行します。
3. 新しい環境に、新しいアプリケーションバージョンをデプロイします。
4. 新しい環境で新しいバージョンをテストします。
5. 環境の概要ページで、[Actions] (アクション) を選択し、[Swap environment URLs] (環境 URL のスワップ) を選択します。
6. [環境名] で、現在の環境を選択します。
7. [Swap] を選択します。

Elastic Beanstalk は古い環境と新しい環境の CNAME レコードを交換して、古いバージョンから新しいバージョンにトラフィックをリダイレクトします。

Elastic Beanstalk がスワップ操作を完了した後、古い環境の URL に接続するときに新しい環境が応答することを検証します。DNS の変更が一括適用され、古い DNS レコードの有効期限が切れるまで、古い環境を終了しないでください。DNS サーバーは、必ずしも DNS レコードで設定した有効期限 (TTL) に基づいてキャッシュから古いレコードを消去するわけではありません。

Elastic Beanstalk 環境との CI/CD 統合の実装

Elastic Beanstalk は、多くの CI/CD ツールと統合して、アプリケーション開発ワークフローを自動化します。CI/CD プラクティスを使用すると、最小限の手動操作でアプリケーションを自動的に構築、テスト、デプロイできます。継続的デリバリー/デプロイ (CD) は、デプロイプロセスを自動化することで継続的インテグレーション (CI) を拡張します。CodePipeline などの AWS サービスや Jenkins や GitLab などのサードパーティーツールを使用して、合理化されたデプロイパイプラインを作成し、Elastic Beanstalk 環境への一貫した信頼性の高いデプロイを確保できます。

AWS 開始するための ソース

次のリストでは、CI/CD ツールと、Elastic Beanstalk 環境への自動デプロイパイプラインを作成するためのstep-by-stepガイダンスを提供する対応する AWS リソースについて説明します。

- AWS CodePipeline – このAWS 入門リソースセンターチュートリアルでは、GitHub 、S3、または から Elastic Beanstalk への継続的デプロイパイプラインを設定する方法を示します AWS CodeCommit。
- GitHub Actions – この .NET on AWS Blog の投稿では、GitHub から直接 Elastic Beanstalk への継続的なデプロイパイプラインを設定するように YAML ベースのワークフローを設定する方法について説明します。
- GitLab – この AWS DevOps 開発者生産性ブログ記事では、GitLab 継続的パイプラインを設定して Node.js アプリケーションを Elastic Beanstalk Docker 環境にデプロイする方法を示します。
- Azure DevOps – この .NET on AWS Blog の投稿では、Azure Pipelines を使用して、Azure DevOps Git リポジトリから Elastic Beanstalk への継続的なデプロイパイプラインを実装する方法について説明します。

その他のリソース

以下のサードパーティーのツールとリソースは、Elastic Beanstalk 環境に自動デプロイパイプラインを実装するのに役立ちます。

- Jenkins – AWS EBDeployment Jenkins プラグインを使用すると、Jenkins ジョブ設定ページから Elastic Beanstalk 環境に直接デプロイできます。
- Circle CI: – Orbs for Elastic Beanstalk は、アプリケーションを Elastic Beanstalk にデプロイおよびスケーリングするための再利用可能な設定パッケージを提供します。
- Bitbucket Pipelines – 記事「Deploy Elastic Beanstalk Application using Bitbucket Pipelines」は、Elastic Beanstalk で Bitbucket Pipelines を実装するための基本的な設定例を提供します。

設定変更

環境マネジメントコンソールの設定セクションで設定オプション設定を変更すると、 は影響を受けるすべてのリソースに変更を AWS Elastic Beanstalk 伝達します。これらのリソースには、お客様のアプリケーションを実行する Amazon EC2 インスタンス、それらのインスタンスを管理する Auto Scaling グループ、EC2 インスタンス自体にトラフィックを分散するロードバランサーが含まれます。

設定変更の多くが、既存のインスタンスを置き換えることなく実行環境に適用できます。たとえば、ヘルスチェック URL を設定すると、環境が更新されロードバランサーの設定が変更されますが、このときダウンタイムは発生しません。これは、アプリケーションを実行しているインスタンスが更新適用中も引き続きリクエストを処理するためです。

起動設定または VPC 設定を変更する設定の変更では、環境内のすべてのインスタンスを終了し置き換える必要があります。たとえば、環境のインスタンスタイプまたは SSH キー設定を変更する場合、EC2 インスタンスを終了し置き換える必要があります。Elastic Beanstalk には、この置き換え方法を決定するいくつかのポリシーが用意されています。

- ローリング更新 – このプロセスでダウンタイムが発生しないように、Elastic Beanstalk はこのような設定の変更をバッチで適用し、常に最低限必要なインスタンスを実行してトラフィックを処理します。この方法により、更新プロセス中のダウンタイムを回避できます。詳細については、「ローリング更新」を参照してください。
- イミュータブルな更新 – Elastic Beanstalk により一時的な Auto Scaling グループが、新しい設定で実行されている別のインスタンスセットで環境外に起動されます。その後、Elastic Beanstalk により、これらのインスタンスが環境のロードバランサーの背後に配置されます。新しいインスタンスがヘルスチェックにパスするまでは、古いインスタンスと新しいインスタンスのいずれもがトラフィックを処理します。この時点で Elastic Beanstalk により、新しいインスタンスが環境の Auto Scaling グループに移動され、一時グループと古いインスタンスが終了します。詳細については、「イミュータブルな更新」を参照してください。
- 無効 – Elastic Beanstalk では、ダウンタイムを回避する対策が行われません。この場合は、環境にある既存のインスタンスが終了し、新しい設定で実行されている新しいインスタンスに置き換えられます。

> **Warning**
>
> 一部のポリシーでは、デプロイ時または更新時にすべてのインスタンスが置き換えられます。これにより、累積したすべての Amazon EC2 バーストバランスが失われます。次の場合に発生します。
>
> - インスタンスの置換を有効にしたマネージドプラットフォームの更新
> - イミュータブルな更新
> - イミュータブルな更新またはトラフィック分割を有効にしたデプロイ

サポートされる更新タイプ

ローリング更新の設定	負荷分散された環境	単一インスタンス環境	レガシー Windows サーバー環境
無効	✓ はい	✓ はい	✓ はい
ヘルスにもとづくローリング	✓ はい	×いいえ	✓ はい
時間にもとづくローリング	✓ はい	×いいえ	✓ はい
Immutable	✓ はい	✓ はい	×いいえ

† このテーブルのレガシー Windows Server 環境は、IIS 8.5 より前の IIS バージョンを使用する Windows Server プラットフォーム設定に基づいています。

トピック

- Elastic Beanstalk 環境設定のローリング更新
- 変更不可能な環境の更新

Elastic Beanstalk 環境設定のローリング更新

設定変更にインスタンスの置き換えが必要な場合、Elastic Beanstalk はバッチ単位で更新を実行することで、変更の適用中のダウンタイムを回避できます。ローリング更新中、容量は 1 バッチのサイズ分だけ減ります。バッチのサイズはお客様が設定できます。Elastic Beanstalk は、停止中のインスタンスのバッチを 1 つ選んで終了させ、新しい設定のバッチを起動します。新しいバッチがリクエストの処理を開始すると、Elastic Beanstalk は次のバッチに進みます。

ローリング設定更新のバッチは、バッチ間に一定の時間間隔を設定して定期的 (時間ベース) に処理することも、状態に基づいて処理することもできます。時間ベースのローリング更新の場合、インスタンスのバッチの起動が完了して、次のバッチに移動するまで Elastic Beanstalk が待機する時間を設定できます。この一時停止の時間によって、アプリケーションはブートストラップおよびリクエスト処理の開始が可能になります。

正常性に基づくローリング更新では、Elastic Beanstalk は、バッチのインスタンスがヘルスチェックに合格するまで待ってから、次のバッチに移ります。インスタンスの状態は、ヘルスレポートシステ

ム (基本または拡張) によって決定されます。基本ヘルスレポートでは、バッチのすべてのインスタンスが Elastic Load Balancing (ELB) ヘルスチェックに合格し、バッチは正常な状態であると見なされます。

拡張ヘルスレポートでは、Elastic Beanstalk が次のバッチを処理する前に、バッチのすべてのインスタンスが複数の連続的なヘルスチェックにパスする必要があります。インスタンスのみを確認する ELB ヘルスチェックに加え、拡張ヘルスは、アプリケーションログと環境のその他のリソースの状態を監視します。拡張ヘルスを使用するウェブサーバー環境では、すべてのインスタンスが 2 分間にわたって行われる 12 のヘルスチェックに合格する必要があります (ワーカー環境では 3 分間にわたって 18 のチェックが行われます) 。いずれかのインスタンスが 1 つでもヘルスチェックで不合格になければ、カウントがリセットされます。

バッチがローリング更新のタイムアウト (デフォルトは 30 分) 以内に正常にならなければ、更新はキャンセルされます。ローリング更新のタイムアウトは、`aws:autoscaling:updatepolicy:rollingupdate` 名前空間で使用できる設定オプションの 1 つです。アプリケーションがヘルスチェックで Ok ステータスにならなくても、別のレベルで安定する場合は、`aws:elasticbeanstalk:healthreporting:system` 名前空間で `HealthCheckSuccessThreshold` オプションを設定して、Elastic Beanstalk でインスタンスが正常と見なされるレベルに変更できます。

ローリング更新プロセスが失敗した場合、Elastic Beanstalk は別のローリング更新を開始して、以前の設定にロールバックします。ローリング更新は、ヘルスチェックに不合格になったことが原因で失敗したり、新しいインスタンスの起動によりアカウントのクォータを超えたことが原因で失敗したりすることがあります。例えば、Amazon EC2 インスタンス数のクォータに達すると、新しいインスタンスのバッチをプロビジョンしようとしたときに、ローリング更新は失敗する可能性があります。この場合、ロールバックも失敗します。

ロールバックが失敗すると、更新プロセスは終了し、環境は異常な状態のままとなります。正常に完了したバッチには新しい設定が反映されますが、未処理のバッチでは依然として古い設定でインスタンスが実行されます。ロールバックに失敗した環境を修復するには、まず更新に失敗した根本的な原因を解決します。次に、あらためて環境の更新を開始します。

代わりの方法として、別の環境にアプリケーションの新しいバージョンをデプロイしてから、CNAME スワップを実行して、ダウンタイムなしでトラフィックをリダイレクトすることがあります。詳細については「Elastic Beanstalk を使用したブルー/グリーンデプロイ」を参照してください。

ローリング更新とローリングデプロイ

ローリング更新が実行されるのは、設定変更に伴い、新しい Amazon EC2 インスタンスをお客様の環境用にプロビジョンすることが必要になるときです。例えば、Auto Scaling グループの設定 (インスタンスタイプとキーペアの設定など) の変更、VPC の設定の変更です。ローリング更新では、インスタンスの各バッチが終了されてから、新しいバッチが置き換え用にプロビジョニングされます。

ローリングデプロイが実行されるのは、毎回、アプリケーションをデプロイするときです。通常、お客様の環境でインスタンスの置き換えなしで実行できます。Elastic Beanstalk は、各バッチをサービスから外し、新しいアプリケーションバージョンをデプロイしてから、サービスに戻します。

ただし例外は、設定の変更に伴い、新しいアプリケーションバージョンをデプロイすると同時にインスタンスを置き換えることが必要になる場合です。たとえば、ソースバンドルにあるキー名設定 (設定ファイル内) を変更し、お客様の環境にデプロイする場合は、ローリング更新がトリガーされます。既存のインスタンスの各バッチに新しいアプリケーションバージョンがデプロイされる代わりに、インスタンスの新しいバッチが新しい設定でプロビジョニングされます。この場合、個別のデプロイは行われません。新しいインスタンスが新しいアプリケーションバージョンで起動されるためです。

新しいインスタンスが環境更新の一部としてプロビジョニングされる場合、アプリケーションのソースコードが新しいインスタンスと、インスタンスが適用されるオペレーティングシステムやソフトウェアを変更するすべての設定にデプロイされるデプロイフェーズがあります。デプロイのヘルスチェック設定 ([Ignore health check] と [Healthy threshold]、[Command timeout]) もデプロイフェーズでローリング更新および変更不可能な更新に基づき、適用されます。

ローリング更新の設定

Elastic Beanstalk コンソールでローリング更新を有効にして設定できます。

ローリング更新を有効にするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [Rolling updates and deployments (ローリング更新とデプロイ)] 設定カテゴリで [Edit (編集)] を選択します。
5. [Configuration updates (設定の更新)] セクションの [ローリング更新のタイプ] で、いずれかの [ローリング] オプションを選択します。

6. [バッチサイズ]、[最小キャパシティー]、[停止時間] の順に選択します。
7. ページの最下部で [適用] を選択し変更を保存します。

[Rolling updates and deployments] (ローリング更新とデプロイ) ページの [Configuration updates] (設定の更新) セクションには、ローリング更新に関する次のオプションがあります。

- Rolling update type – Elastic Beanstalk が、インスタンスのバッチの更新を終え、次のバッチに進むまで待機し、その間インスタンスがブートストラップを終了して、トラフィック処理を開始できるようになります。次のオプションから選択します。
  - Rolling based on Health – 現在のバッチのインスタンスが正常な状態になるまで待ってから、インスタンスを実行状態にし、次のバッチを処理します。
  - Rolling based on Time – 新しいインスタンスを開始して実行状態にし、次のバッチを処理するまでの待機時間を指定します。
  - Immutable – 変更不可能な更新を実行することで、設定変更をインスタンスの新しいグループに適用します。
- Batch size – 各バッチで置き換えるインスタンスの数です。**`1`** ~ **`10000`** のいずれかの値を指定できます。デフォルトでは、この値は Auto Scaling グループの最小サイズの 1/3 を整数に切り上げた値になります。
- Minimum capacity – 他のインスタンスの更新中に実行され続けるインスタンスの最小数。**`0`** ~ **`9999`** のいずれかの値を指定できます。デフォルト値は Auto Scaling グループの最小サイズ、または Auto Scaling グループの最大サイズ未満の値のいずれかの低い数値になります。
- Pause time (時間ベースのみ) - バッチが更新されてから次のバッチに移るまでに待つ時間。その間に、アプリケーションがトラフィックの受信を開始できるようになります。0 秒 ~ 1 時間のいずれかの値を指定できます。

aws:autoscaling:updatepolicy:rollingupdate 名前空間

また、`aws:autoscaling:updatepolicy:rollingupdate` 名前空間にある設定オプションを利用してローリング更新を設定できます。

`RollingUpdateEnabled` オプションを使用してローリング更新を有効にし、更新タイプの選択に `RollingUpdateType` を使用します。次の値は、`RollingUpdateType` をサポートします。

- [`Health`] – 現在のバッチのインスタンスが正常な状態になるまで待ってから、インスタンスを実行状態にし、次のバッチを処理します。

- Time – 新しいインスタンスを起動して、サービス実行状態にし、次のバッチを開始するまでの待機時間を指定します。
- Immutable – 変更不可能な更新を実行することで、設定変更をインスタンスの新しいグループに適用します。

ローリング更新を有効にした場合は、MaxBatchSize オプションと MinInstancesInService オプションも設定してバッチのサイズを設定します。時間ベースと正常性ベースのローリング更新の場合、それぞれ PauseTime および Timeout をそれぞれ設定できます。

たとえば、一度に最大 5 つのインスタンスを起動する場合、少なくとも 2 つのインスタンスを実行状態に維持したままバッチごとに 5 分 30 秒間待機し、次のオプションと値を指定します。

Example .ebextensions/timebased.config

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateEnabled: true
    MaxBatchSize: 5
    MinInstancesInService: 2
    RollingUpdateType: Time
    PauseTime: PT5M30S
```

正常性に基づくローリング更新を有効にするには、各バッチにつき 45 分間のタイムアウトで次のオプションと値を指定します。

Example .ebextensions/healthbased.config

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateEnabled: true
    MaxBatchSize: 5
    MinInstancesInService: 2
    RollingUpdateType: Health
    Timeout: PT45M
```

[Timeout] と [PauseTime] の値は、ISO8601 duration: PT#H#M#S で指定する必要があります。ここで、各 # はそれぞれ時間、分、秒を指します。

EB CLI および Elastic Beanstalk コンソールでは、上記のオプションに推奨値が適用されます。設定ファイルを使用して同じファイルを設定する場合は、これらの設定を削除する必要があります。詳細については、「推奨値」を参照してください。

変更不可能な環境の更新

変更不可能な更新はローリング更新の代替手段です。変更不可能な環境の更新により、インスタンスの置換を必要とする構成変更が効率的かつ安全に適用されます。変更不可能な環境の更新に失敗した場合のロールバックプロセスは、Auto Scaling グループの終了のみです。しかし、ローリーング更新に失敗した場合は、追加のローリング更新を実行して変更を元に戻さなければなりません。

変更不可能な環境の更新を実行するために、Elastic Beanstalk は環境のロードバランサーの背後に 2 つ目の (一時的な) Auto Scaling グループを作成し、新しいインスタンスを作成します。これにより、Elastic Beanstalk は新しいグループ内の新規設定で単一のインスタンスを起動します。このインスタンスは、以前の設定を実行している元の Auto Scaling グループの全インスタンスと共にトラフィックを処理します。

最初のインスタンスがヘルスチェックに合格すると、Elastic Beanstalk は元の Auto Scaling グループで実行しているインスタンスの数に一致する新しい設定で、追加のインスタンスを起動します。新しいインスタンスがすべてヘルスチェックに合格した場合、Elastic Beanstalk はこれらのインスタンスを元の Auto Scaling グループに転送し、一時 Auto Scaling グループと古いインスタンスを終了します。

> **Note**
>
> 変更不可能な環境の更新中、新しい Auto Scaling グループ内のインスタンスがリクエスト処理を開始してから、元の Auto Scaling グループのインスタンスが終了するまでの短い間、環境の容量が倍増します。環境内に多くのインスタンスがある場合、あるいはオンデマンドインスタンスクォータが低い場合は、変更不可能な環境の更新を実行する上で必要な容量が十分あることを確認してください。クォータの下限に近い場合は、代わりにローリング更新を行うことをお勧めします。

変更不可能な更新では、更新中に環境の状態を評価する拡張ヘルスレポートが必要となります。拡張ヘルスレポートは、ロードバランサーの標準ヘルスチェックと組み合わせることで、新しい設定を実行しているインスタンスがリクエスト処理を正しく実行していることを確認します。

また、ローリングデプロイを実行する代わりに、変更不可能な更新で新しいアプリケーションのバージョンをデプロイすることも可能です。変更不可能な更新でアプリケーションのデプロイを実行する

よう Elastic Beanstalk を設定すると、新しいアプリケーションのバージョンをデプロイするたびに環境内にあるすべてのインスタンスが置き換えられます。変更不可能なアプリケーションのデプロイが失敗した場合、Elastic Beanstalk は新しい Auto Scaling グループを終了して直ちに変更を元に戻します。これは、部分的なフリートのデプロイを防止するためです。部分的なフリートのデプロイは、一部のバッチしか終了していない状態でローリングデプロイに失敗すると発生します。

> **Warning**
>
> 一部のポリシーでは、デプロイ時または更新時にすべてのインスタンスが置き換えられます。これにより、累積したすべての Amazon EC2 バーストバランスが失われます。次の場合に発生します。
>
> - インスタンスの置換を有効にしたマネージドプラットフォームの更新
> - イミュータブルな更新
> - イミュータブルな更新またはトラフィック分割を有効にしたデプロイ

変更不可能な更新に失敗した場合、Elastic Beanstalk がインスタンスを削除する前に、新しいインスタンスによりバンドルログが Amazon S3 にアップロードされます。Elastic Beanstalk では、失敗した更新のログを Amazon S3に 1 時間保管してから削除します(バンドルログやログ末尾の場合は標準の 15 分)。

> **Note**
>
> 変更不可能な更新で設定を適用せずにアプリケーションバージョンをデプロイした場合、ローリング更新をトリガーする設定の変更 (インスタンスタイプの変更設定など) を含むアプリケーションバージョンをデプロイしようとするとエラーが生じる可能性があります。こうしたエラーを防ぐには、別の更新で設定を変更するか、デプロイと設定変更の両方に対して変更不可能な更新を設定します。

変更不可能な更新は、リソースの設定変更と共に実行することはできません。たとえば、他の設定を更新している間はインスタンス置換を必要とする設定を変更したり、設定やソースコード内の追加リソースを変更する設定ファイルと共に変更不可能なデプロイを実行することはできません。リソース設定 (ロードバランサーの設定など) を変更し、変更不可能な更新を同時に実行すると、Elastic Beanstalk によりエラーが返されます。

ソースコードやインスタンス設定の変更に依存しないリソース設定の変更については、2 回に分けて更新します。依存する場合は、青/緑のデプロイを実行してください。

変更不可能な更新を設定する

Elastic Beanstalk コンソールで変更不可能な更新を有効にして設定できます。

変更不可能な更新を有効にする (コンソール)

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [Rolling updates and deployments (ローリング更新とデプロイ)] 設定カテゴリで [Edit (編集)] を選択します。
5. [設定の更新] セクションで [ローリング更新のタイプ] を [Immutable (変更不可)] に設定します。
6. ページの最下部で [適用] を選択し変更を保存します。

aws:autoscaling:updatepolicy:rollingupdate 名前空間

`aws:autoscaling:updatepolicy:rollingupdate`名前空間にあるオプションを使用して変更不可能な更新を設定することも可能です。次の設定ファイルは、設定変更に対する変更不可能な更新を有効にします。

Example .ebextensions/immutable-updates.config

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Immutable
```

次の例は、設定変更とデプロイの両方に対する変更不可能な更新を有効にします。

Example .ebextensions/immutable-all.config

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Immutable
  aws:elasticbeanstalk:command:
    DeploymentPolicy: Immutable
```

EB CLI および Elastic Beanstalk コンソールでは、上記のオプションに推奨値が適用されます。設定ファイルを使用して同じファイルを設定する場合は、これらの設定を削除する必要があります。詳細については、「推奨値」を参照してください。

Elastic Beanstalk 環境のプラットフォームバージョンの更新

Elastic Beanstalk は、新しいプラットフォームバージョンを定期的にリリースして、すべての Linux ベースおよび Windows Server ベースの プラットフォームを更新します。新しいプラットフォームバージョンは、既存のソフトウェアコンポーネントへの更新と、新しい機能および設定オプションのサポートを提供します。プラットフォームとプラットフォームバージョンの詳細については、「Elastic Beanstalk プラットフォームの用語集」を参照してください。

Elastic Beanstalk コンソールまたは EB CLI を使用して環境のプラットフォームバージョンを更新できます。更新先のプラットフォームバージョンに応じて、Elastic Beanstalk ではプラットフォームの更新を実行する 2 つの方法のいずれかを推奨します。

- 方法 1 – 環境のプラットフォームバージョンを更新する。この方法は、同一のプラットフォームブランチ (同じランタイム、ウェブサーバー、アプリケーションサーバー、およびオペレーティングシステムを使用) の最新のプラットフォームバージョンに更新する場合で、メジャープラットフォームバージョンを変更しない場合にお勧めします。これは、最も一般的で定期的なプラットフォームの更新です。
- 方法 2 – Blue/Green デプロイを実行する。この方法は、異なるプラットフォームブランチ (異なるランタイム、ウェブサーバー、アプリケーションサーバー、またはオペレーティングシステムを使用) のプラットフォームバージョンに更新する場合、または異なるメジャープラットフォームバージョンに更新する場合にお勧めします。これは、新しいランタイム機能や最新の Elastic Beanstalk 機能を利用する場合、あるいは非推奨または廃止されたプラットフォームブランチから移行する場合に適しています。

  Linux プラットフォームバージョンから移行するには、青/緑のデプロイが必要です。これらのプラットフォームバージョンは現在サポートされているバージョンと互換性がないためです。

  Amazon Linux 2 プラットフォームバージョンは、以前の Amazon Linux AMI プラットフォームバージョンと互換性がないため、Linux アプリケーションを Amazon Linux 2 に移行するには、Blue/Green デプロイが必要です。

プラットフォームの最適な更新方法の選択に関する詳細については、環境のプラットフォームに関するセクションを展開してください。

Docker

方法 1 を使用してプラットフォームの更新を行います。

複数コンテナの Docker

方法 1 を使用してプラットフォームの更新を行います。

事前設定済み Docker

以下の場合を考慮します。

- アプリケーションを別のプラットフォームに移行させる場合。例えば、Go 1.4 (Docker) から Go 1.11 に移行させたり、Python 3.4 (Docker) から Python 3.6 に移行させたりする場合は、方法 2 を使用します。
- アプリケーションを別の Docker コンテナバージョンに移行させる場合。例えば、Glassfish 4.1 (Docker) から Glassfish 5.0 (Docker) に移行させる場合は、方法 2 を使用します。
- 最新のプラットフォームバージョンに更新する際に、コンテナバージョンやメジャーバージョンの変更がない場合は、方法 1 を使用します。

Go

方法 1 を使用してプラットフォームの更新を行います。

Java SE

以下の場合を考慮します。

- アプリケーションを別の Java ランタイムバージョンに移行させる場合。例えば、Java 7 から Java 8 に移行させる場合は、方法 2 を使用します。
- 最新のプラットフォームバージョンに更新する際に、ランタイムバージョンの変更がない場合は、方法 1 を使用します。

Java と Tomcat

以下の場合を考慮します。

- アプリケーションを別の Java ランタイムバージョンまたは Tomcat アプリケーションサーバーバージョンに移行させる場合。例えば、Java 7 と Tomcat 7 から Java 8 と Tomcat 8.5 に移行させる場合は、方法 2 を使用します。
- アプリケーションを Java と Tomcat のメジャープラットフォームバージョン (v1.x.x、v2.x.x、v3.x.x) 間で移行させる場合は、方法 2 を使用します。
- 最新のプラットフォームバージョンに更新する際に、ランタイムバージョン、アプリケーションサーバーバージョン、またはメジャーバージョンに変更がない場合は、方法 1 を使用します。

IIS を使用する Windows Server での .NET

以下の場合を考慮します。

- アプリケーションを別の Windows オペレーティングシステムバージョンに移行させる場合。例えば、Windows Server 2008 R2 から Windows Server 2016 に移行させる場合は、方法 2 を使用します。
- アプリケーションを Windows Server のメジャープラットフォームバージョン間で移行させる場合は、「Windows サーバープラットフォームの以前のメジャーバージョンからの移行」を参照し、方法 2 を使用します。
- アプリケーションが現在 Windows Server プラットフォーム V2.x.x で実行されていて、最新のプラットフォームバージョンに更新する場合は、方法 1 を使用します。

> **Note**
>
> Windows Server プラットフォームバージョンが v2 より前である場合は、意味論的にバージョン管理されません。これらの Windows Server メジャープラットフォームバージョンについては、それぞれの最新バージョンのみを起動できます。また、アップグレード後のロールバックはできません。

Node.js

方法 2 を使用してプラットフォームの更新を行います。

PHP

以下の場合を考慮します。

- アプリケーションを別の PHP ランタイムバージョンに移行させる場合。例えばPHP 5.6 から PHP 7.2 に移行させる場合は、方法 2 を使用します。
- アプリケーションを PHP のメジャープラットフォームバージョン (v1.x.x と v2.x.x) 間で移行させる場合は、方法 2 を使用します。
- 最新のプラットフォームバージョンに更新する際に、ランタイムバージョンやメジャーバージョンの変更がない場合は、方法 1 を使用します。

Python

以下の場合を考慮します。

- アプリケーションを別の Python ランタイムバージョンに移行させる場合。例えば、Python 2.7 から Python 3.6 に移行させる場合は、方法 2 を使用します。
- アプリケーションを Python のメジャープラットフォームバージョン (v1.x.x と v2.x.x) 間で移行させる場合は、方法 2 を使用します。
- 最新のプラットフォームバージョンに更新する際に、ランタイムバージョンやメジャーバージョンの変更がない場合は、方法 1 を使用します。

Ruby

以下の場合を考慮します。

- アプリケーションを別の Ruby ランタイムバージョンまたはアプリケーションサーバーバージョンに移行させる場合。例えば、Ruby 2.3 と Puma から Ruby 2.6 と Puma に移行させる場合は、方法 2 を使用します。
- アプリケーションを Ruby のメジャープラットフォームバージョン (v1.x.x と v2.x.x) 間で移行させる場合は、方法 2 を使用します。
- 最新のプラットフォームバージョンに更新する際に、ランタイムバージョン、アプリケーションサーバーバージョン、またはメジャーバージョンに変更がない場合は、方法 1 を使用します。

方法 1 – 環境のプラットフォームバージョンを更新する

この方法を使用して、環境のプラットフォームブランチを最新バージョンに更新します。以前のプラットフォームバージョンを使用して環境を作成した場合や、古いバージョンから環境をアップグレードした場合は、この方法を使用して、同じプラットフォームブランチにある以前のプラットフォームバージョンに戻すこともできます。

環境のプラットフォームを更新するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページの [プラットフォーム] で、[変更] を選択します。
4. [プラットフォームのバージョンの更新] ダイアログで、プラットフォームのバージョンを選択します。ブランチ内の最新 (推奨) のプラットフォームバージョンが自動的に選択されます。過去に使用したことがあるどのバージョンにも更新できます。
5. [Save] を選択します。

さらにプラットフォームの更新を簡素化するために、Elastic Beanstalk で自動的に管理することもできます。環境を設定して、週ごとのメンテナンス時間にマイナーバージョンとパッチバージョンの更新が自動的に適用されるようできます。Elastic Beanstalk では、マネージド更新がダウンタイムや容量の減少なしに適用されます。また、この新バージョンでアプリケーションを実行しているインスタンスがヘルスチェックにパスしなかった場合、マネージド更新はすぐにキャンセルされます。詳細については、「マネージドプラットフォーム更新」を参照してください。

方法 2 – Blue/Green デプロイを実行する

この方法を使用して、異なるプラットフォームブランチ (異なるランタイム、ウェブサーバー、アプリケーションサーバー、またはオペレーティングシステムを使用)、または異なるメジャープラットフォームバージョンにアップデートします。これは、通常、新しいランタイム機能や最新の Elastic Beanstalk 機能を利用する場合に必要です。また、非推奨あるいは廃止されたプラットフォームブランチから移行する場合にも必要です。

メジャープラットフォームバージョン間で移行したり、コンポーネントのメジャー更新を伴うプラットフォームバージョンに移行したりする場合は、アプリケーションやその一部が新しいプラットフォームバージョンで正常に機能しないことがあり、変更が必要になる場合があります。

移行を実行する前に、ローカル開発マシンを、移行先のプラットフォームの最新のランタイムバージョンや他のコンポーネントに更新します。アプリケーションが正常に機能することを確認し、必要なコードの修正や変更を行います。次に、以下のベストプラクティス手順に従って環境を新しいプラットフォームバージョンに安全に移行させます。

メジャー更新を伴うプラットフォームバージョンに環境を移行させるには

1. 新しいターゲットプラットフォームバージョンを使用して新しい環境を作成し、アプリケーションコードをその環境にデプロイします。新しい環境は、移行する環境がある Elastic Beanstalk アプリケーションに含まれている必要があります。既存の環境はまだ終了しないでください。
2. アプリケーションコードを移行する先の新しい環境を使用します。特に、次のことに注意してください。

   - 開発フェーズで検出できなかったアプリケーションの互換性問題を見つけて修正します。
   - アプリケーションで設定ファイルを使用して行ったすべてのカスタマイズが新しい環境で正しく機能することを確認します。これには、オプション設定、追加でインストールしたパッケージ、カスタムセキュリティポリシー、環境インスタンスにインストールしたスクリプトや設定ファイルが含まれる場合があります。
   - アプリケーションでカスタム Amazon Machine Image (AMI) を使用している場合は、新しいプラットフォームバージョンの AMI に基づいて新しいカスタム AMI を作成します。詳細については、「Elastic Beanstalk 環境でのカスタム Amazon マシンイメージ (AMI) の使用」を参照してください。特に、これが必要となるのは、アプリケーションで使用している Windows Server プラットフォームに AMI が含まれていて、Windows Server V2 プラットフォームバージョンに移行する場合です。この場合は、「Windows サーバープラットフォームの以前のメジャーバージョンからの移行」も参照してください。

   新しい環境でアプリケーションが適切に動作することが確認されるまで、修正のテストとデプロイを繰り返します。
3. CNAME を既存の本稼働環境の CNAME と交換することにより、新しい環境を本稼働環境に切り替えます。詳細については、「Elastic Beanstalk を使用したブルー/グリーンデプロイ」を参照してください。
4. 本稼働環境の新しい環境の状態が適切であることを確認したら、古い環境を終了します。詳細については、Elastic Beanstalk 環境を終了する を参照してください

マネージドプラットフォーム更新

AWS Elastic Beanstalk は、プラットフォームの更新を定期的にリリースして、修正、ソフトウェアの更新、新機能を提供します。マネージドプラットフォーム更新機能により、予定済みのメンテナンス期間中に、環境を自動的に最新バージョンのプラットフォームに更新できます。更新プロセス中も、アプリケーションは能力を低減させることなく稼動状態に保たれます。マネージド更新は、単一インスタンス環境とロードバランシング環境の両方で利用できます。

> **Note**
>
> この機能は、バージョン 2 (v2) 以前の Windows Server プラットフォームバージョンでは使用できません。

環境に、自動的にパッチバージョンの更新またはパッチとマイナーバージョンの更新の両方を適用するよう設定できます。マネージドプラットフォームの更新では、プラットフォームブランチ全体にわたる更新 (オペレーティングシステム、ランタイム、Elastic Beanstalk コンポーネントなどのプラットフォームコンポーネントの異なるメジャーバージョンへの更新) はサポートされません。これは、下位互換性のない変更が導入される可能性があるためです。

プラットフォーム更新が利用できない場合でもメンテナンス期間中に環境のすべてのインスタンスを置換するよう Elastic Beanstalk を設定できます。環境にあるすべてのインスタンスを置換することは、アプリケーションを長期的に稼働させる際に際に発生するバグやメモリ問題の解決に有効です。

Elastic Beanstalk コンソールを使用して 2019 年 11 月 25 日以降に作成された環境では、メネージドアップデートは可能な限りデフォルトで有効になります。管理された更新を有効にするには、拡張ヘルスが必要です。拡張ヘルスは、設定プリセットの 1 つを選択するとデフォルトで有効になり、[Custom configuration (カスタム設定)] を選択すると無効になります。拡張ヘルスをサポートしていない古いバージョンのプラットフォームの場合、または拡張ヘルスが無効になっている場合は、コンソールは管理された更新を有効にできません。コンソールで新しい環境で管理された更新を有効にすると、[Weekly update window (週次更新ウィンドウ)] はランダムな時刻にランダムな曜日に設定されます。[Update level (更新レベル)] が [Minor and patch (マイナーおよびパッチ)] に設定され、[Instance replacement (インスタンスの置き換え)] が無効になります。最終的な環境作成ステップの前に、管理された更新を無効化または再設定できます。

既存の環境の場合は、いつでも Elastic Beanstalk コンソールを使用して、マネージドプラットフォームの更新を設定します。

> **Important**
>
> 1つの AWS アカウント内の Beanstalk 環境の数値が高い場合、管理された更新中のスロットリングの問題のリスクを示していることがあります。高い数値は、環境に対する管理された更新のスケジュールの密度に基づく相対的な量です。1 つのアカウントで 200 を超える環境が密にスケジュールされている場合、スロットリングの問題が発生する可能性がありますが、その数が少ない場合も問題になる可能性があります。
> 管理された更新のリソース負荷を分散するために、1 つのアカウント内の環境のスケジュールされたメンテナンスウィンドウを分散させることをお勧めします。

また、マルチアカウント戦略を検討してください。詳細については、ホワイトペーパーとガイドのウェブサイトの「複数のアカウントを使用した AWS 環境の整理」を参照してください。 AWS

マネージドプラットフォームの更新を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [管理された更新] カテゴリで、[編集] を選択します。
5. [Managed updates (管理された更新)] を無効または有効にします。
6. 管理された更新が有効になっている場合は、メンテナンスウィンドウを選択し、[Update level (更新レベル)] を選択します。
7. (オプション) [Instance replacement (インスタンスの置換)] を選択して毎週のインスタンス置換を有効にします。

8. ページの最下部で [適用] を選択し変更を保存します。

マネージドプラットフォーム更新において、アプリケーションが正常でプラットフォーム更新が問題なく実行されたかどうかの判断は、拡張ヘルスレポートに依存しています。手順については、「Elastic Beanstalk の拡張ヘルスレポートの有効化」を参照してください。

セクション

- マネージドプラットフォーム更新を実行するために必要なアクセス許可
- マネージド更新のメンテナンス期間
- マイナーバージョンとパッチバージョンの更新
- 変更不可能な環境の更新
- マネージド更新の管理
- マネージドアクションオプションの名前空間

マネージドプラットフォーム更新を実行するために必要なアクセス許可

Elastic Beanstalk には、ユーザーに代わりプラットフォーム更新を実行するためのアクセス許可が必要です。これらのアクセス許可を得るために、Elastic Beanstalk ではマネージド更新サービスロールが使用されます。環境に対してデフォルトのサービスロールを使用すると、Elastic Beanstalk コンソールではマネージド更新サービスロールとしても使用されます。コンソールによって、`AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` 管理ポリシーがサービスロールに割り当てられます。このポリシーには、マネージドプラットフォーム更新を実行するために Elastic Beanstalk が必要とするすべてのアクセス許可が含まれています。

マネージド更新サービスロールを設定するその他の方法の詳細については、「the section called “サービス役割”」を参照してください。

> **Note**
>
> 追加のリソースを含めるために環境を拡張する設定ファイルを使用している場合は、環境のマネージド更新サービスロールにアクセス許可を追加する必要があります。通常、他のセクションまたはファイルでこれらのリソースを名前で参照する場合、権限を追加する必要があります。

更新に失敗した場合、失敗の理由は [マネージド更新] ページで確認できます。

マネージド更新のメンテナンス期間

が環境のプラットフォームの新しいバージョンを AWS リリースすると、Elastic Beanstalk は、次の毎週のメンテナンスウィンドウ中にマネージドプラットフォームの更新をスケジュールします。メンテナンス期間は 2 時間です。Elastic Beanstalk は、スケジュールされた更新をメンテナンス期間中に開始します。更新はこの期間が終了するまで完了しない場合があります。

> **Note**
>
> ほとんどの場合、Elastic Beanstalk は翌週のメンテナンス期間中にマネージド更新が実行されるようスケジュールします。このシステムは、管理された更新のスケジュールにおいて更新の安全性およびサービスの可用性に関するさまざまな側面を考慮します。まれに、初めてのメンテナンス期間には更新がスケジュールされないことがあります。この状態が発生した場合、システムは次のメンテナンス期間中に再度試行します。マネージド更新を手動で適用

するには、このページの「マネージド更新の管理」で説明するように、[Apply now (今すぐ適用)] を選択します。

マイナーバージョンとパッチバージョンの更新

マネージドプラットフォーム更新がパッチバージョンの更新のみ、またはマイナーバージョンとパッチバージョンの更新の両方に適用されるよう設定できます。パッチバージョンの更新には、バグ修正およびパフォーマンスの改善点をはじめ、インスタンス上のソフトウェアやスクリプト、設定オプションに対するマイナーな変更点も含まれる可能性があります。マイナーバージョンの更新では、Elastic Beanstalk の新機能へのサポートを提供します。マネージドプラットフォーム更新で、後方互換性のない変更を行うメジャーバージョンの更新を適用することはできません。

プラットフォームのバージョン番号で、2 番目の数はマイナーバージョンの更新、3 番目の数はパッチバージョンの更新を意味します。たとえば、バージョン 2.0.7 プラットフォームの場合、マイナーバージョンは 0、パッチバージョンは 7 です。

変更不可能な環境の更新

マネージドプラットフォーム更新は、変更不可能な環境の更新を実行し、環境を新しいプラットフォームバージョンにアップグレードします。変更不可能な更新によって、新しいバージョンを実行しているインスタンスがヘルスチェックにパスしたかどうかを確認する前に、インスタンスをサービス停止状態にしたり、変更を加えることもなく環境を更新します。

変更不可能な更新では、Elastic Beanstalk は新しいプラットフォームバージョンで現在実行しているインスタンスと同数のインスタンスをデプロイします。新しいインスタンスは、古いバージョンを起動しているインスタンスとともにリクエスト取得を開始します。新しいインスタンスセットがすべてのヘルスチェックに合格すると、Elastic Beanstalk は新しいバージョンのインスタンスのみを残して、古いインスタンスセットを終了させます。

マネージドプラットフォーム更新は、メンテナンス期間外に適用したとしても、常に変更不可能な更新を実行します。[ダッシュボード] からプラットフォームのバージョンを変更すると、Elastic Beanstalk は設定更新のために選択した更新ポリシーを適用します。

> **Warning**
>
> 一部のポリシーでは、デプロイ時または更新時にすべてのインスタンスが置き換えられます。これにより、累積したすべての Amazon EC2 バーストバランスが失われます。次の場合に発生します。

- インスタンスの置換を有効にしたマネージドプラットフォームの更新
- イミュータブルな更新
- イミュータブルな更新またはトラフィック分割を有効にしたデプロイ

マネージド更新の管理

Elastic Beanstalk コンソールには、[マネージド更新の概要] ページの管理された更新に関する詳細情報が表示されます。

マネージド更新（コンソール）に関する情報を表示するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [管理された更新] を選択します。

[マネージド更新の概要] セクションは、予定済みおよび保留中のマネージド更新に関する情報を提供します。[History] セクションには、成功および失敗した更新がリストされます。

予定済みの更新を、メンテナンス期間を待つことなく、即座に適用する選択が可能です。

マネージドプラットフォーム更新（コンソール）を即座に適用するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [管理された更新] を選択します。
4. [Apply] を選択します。
5. 更新の詳細を確認し、[適用] を選択します。

マネージドプラットフォーム更新をメンテナンス期間外に適用すると、Elastic Beanstalk は変更不可能な更新を実行します。環境のプラットフォームをダッシュボードまたは他のクライアントを使って更新すると、Elastic Beanstalk は設定の変更のために選択した更新タイプを使用します。

予定されたマネージド更新がない場合は、環境がすでに最新バージョンを実行している可能性があります。予定された更新がないその他の理由は、次のとおりです。

- マイナーバージョン更新が利用可能であるが、環境は自動的にパッチバージョン更新のみを適用するよう設定されている。
- 環境が更新のリリース以後、スキャンされていない。Elastic Beanstalk は通常、1 時間ごとに更新を確認する。
- 更新が保留中またはすでに進行中である。

メンテナンス期間が開始されている、または [今すぐ適用] が選択されており、予定された更新が実行前に保留状態になっている。

マネージドアクションオプションの名前空間

`aws:elasticbeanstalk:managedactions` および `aws:elasticbeanstalk:managedactions:platformupdate` 名前空間にある設定オプションを使って、マネージドプラットフォーム更新を有効化し、設定できます。

`ManagedActionsEnabled` オプションは、マネージドプラットフォーム更新をオンにします。このオプションを `true` に設定するとマネージドプラットフォーム更新が有効になり、他のオプションを使用すると更新動作を設定できます。

`PreferredStartTime` を使用して、毎週のメンテナンス期間の開始時刻を、`#:##:#`形式で設定します。

`UpdateLevel` を `minor` または `patch` に設定すると、マイナーおよびパッチバージョン更新の両方、あるいはパッチバージョンの更新のみを適用するようそれぞれ設定できます。

マネージドプラットフォーム更新を有効にしている場合、`InstanceRefreshEnabled` オプションを `true` に設定することでインスタンス置換も有効にできます。この設定が有効になっている場合、プラットフォームの新しいバージョンが利用可能かどうかに関係なく、Elastic Beanstalk は毎週環境に対する変更不可能な更新を実行します。

次の例は、毎週火曜日の午前 9 時 (UTC) にメンテナンス期間が開始され、パッチバージョン更新がマネージドプラットフォーム更新で有効化されることを示す、設定ファイルです。

Example .ebextensions/managed-platform-update.config

```
option_settings:
  aws:elasticbeanstalk:managedactions:
    ManagedActionsEnabled: true
    PreferredStartTime: "Tue:09:00"
  aws:elasticbeanstalk:managedactions:platformupdate:
```

```
    UpdateLevel: patch
    InstanceRefreshEnabled: true
```

レガシーのプラットフォームバージョンからアプリケーションを移行する

レガシーのプラットフォームバージョンを使用する Elastic Beanstalk アプリケーションをデプロイした場合は、新しい機能にアクセスできるよう、レガシーではないプラットフォームバージョンを使用してアプリケーションを新しい環境に移行してください。アプリケーションの実行にレガシープラットフォームバージョンを使用しているかどうか不明な場合は、Elastic Beanstalk コンソールで確認できます。手順については、「レガシープラットフォームバージョンを使っているかどうかを調べるには」を参照してください。

レガシープラットフォームバージョンにはどのような新しい機能が抜けていますか？

レガシーのウェブプラットフォームは次の機能をサポートしません。

- 設定ファイル (.ebextensions) による高度な環境のカスタマイズ トピックで説明した設定ファイル
- ベーシックヘルスレポート トピックで説明した ELB の健康チェック
- Elastic Beanstalk インスタンスプロファイルの管理 トピックで説明したインスタンスプロファイル
- Amazon VPC で Elastic Beanstalk を使用する トピックで説明した VPC
- Elastic Beanstalk 環境にデータベースを追加する トピックで説明したデータ層
- Elastic Beanstalk ワーカー環境 トピックで説明したワーカー層
- 環境タイプ トピックで説明した単一インスタンス環境
- Elastic Beanstalk 環境でのリソースのタグ付け トピックで説明したタグ
- Elastic Beanstalk 環境設定のローリング更新 トピックで説明したローリング更新

一部のプラットフォームバージョンがレガシーとマークされているのはなぜですか?

一部の古いプラットフォームのバージョンでは、最新の Elastic Beanstalk 機能はサポートされていません。これらのバージョンは Elastic Beanstalk コンソールの環境概要ページで [(legacy) (レガシー)] とマークされます。

レガシープラットフォームバージョンを使っているかどうかを調べるには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[プラットフォーム] 名を表示します。

   プラットフォーム名の横に [(legacy) (レガシー)] が表示されている場合、アプリケーションでレガシープラットフォームバージョンを使用しています。

アプリケーションを移行するには

1. アプリケーションを新しい環境にデプロイするには 方法については、「Elastic Beanstalk 環境の作成」を参照してください。
2. Amazon RDS DB インスタンスが存在する場合は、新しい環境の EC2 セキュリティグループへのアクセスを許可するため、データベースセキュリティグループを更新してください。 AWS マネジメントコンソールを使用して EC2 セキュリティグループの名前を検索する方法については、「」を参照してくださいEC2 セキュリティグループ。EC2 セキュリティグループの詳しい設定方法については、Amazon Relational Database Service ユーザーガイドの「DB セキュリティグループの操作」にある「Amazon EC2 セキュリティグループへのネットワークアクセスの許可」セクションをご覧ください。
3. 環境 URL をスワップします。方法については、「Elastic Beanstalk を使用したブルー/グリーンデプロイ」を参照してください。
4. 以前の環境を終了します。方法については、「Elastic Beanstalk 環境を終了する」を参照してください。

> **Note**
>
> AWS Identity and Access Management (IAM) を使用する場合は、 AWS CloudFormation と Amazon RDS (該当する場合) を含めるようにポリシーを更新する必要があります。詳細については、「での Elastic Beanstalk の使用 AWS Identity and Access Management」を参照してください。

Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する

このセクションでは、以下の移行パスのいずれかを使用してアプリケーションを移行する方法について説明します。

- Amazon Linux 2 プラットフォームブランチから Amazon Linux 2023 プラットフォームブランチに移行する。
- Amazon Linux AMI (AL1) プラットフォームブランチから Amazon Linux 2023 (推奨) または Amazon Linux 2 プラットフォームブランチに移行する。

トピック

- Amazon Linux 2 から Amazon Linux 2023 への移行
- Amazon Linux AMI (AL1) から AL2 または AL2023 への移行

Amazon Linux 2 から Amazon Linux 2023 への移行

このトピックでは、Amazon Linux 2 プラットフォームブランチから Amazon Linux 2023 プラットフォームブランチにアプリケーションを移行するためのガイダンスを提供します。

相違点と互換性

Elastic Beanstalk AL2 および AL2023 プラットフォーム間

Elastic Beanstalk Amazon Linux 2 プラットフォームと Amazon Linux 2023 プラットフォームの間には高度な互換性があります。ただし、留意すべき違いがいくつかあります。

- インスタンスメタデータサービスバージョン 1 (IMDSv1) – AL2023 プラットフォームでは、DisableIMDSv1 オプション設定がデフォルトで `true` に設定されます。デフォルトは AL2 プラットフォーム上の `false` です。
- pkg-repo インスタンスツール – pkg-repo ツールは、AL2023 プラットフォームで実行されている環境では使用できません。ただし、パッケージとオペレーティングシステムの更新を AL2023 インスタンスに手動で適用することはできます。詳細については、「Amazon Linux 2023 ユーザーガイド」の「パッケージとオペレーティングシステムの更新の管理」を参照してください。
- Apache HTTPd 設定 – AL2023 プラットフォームの Apache `httpd.conf` ファイルには、AL2 の構成設定とは異なるいくつかの構成設定があります。
  - デフォルトでは、サーバーのファイルシステム全体へのアクセスを拒否します。これらの設定については、Apacheウェブサイトの「セキュリティのヒント」ページの「デフォルトでサーバーファイルを保護する」で説明されています。
  - 設定したセキュリティ機能をユーザーが上書きできないようにします。この設定では、特別に有効になっているディレクトリを除き、すべてのディレクトリの `.htaccess` の設定へのアクセスが拒否されます。この設定については、Apache ウェブサイトの「セキュリティのヒント」

ページの「システム設定の保護」で説明されています。「Apache HTTP サーバーチュートリアル: .htaccess ファイル」ページには、この設定がパフォーマンスの改善に役立つ可能性がある旨が記載されています。

- 名前パターン `.ht*` のファイルへのアクセスを拒否します。この設定により、ウェブクライアントは `.htaccess` および `.htpasswd` ファイルを表示できなくなります。

上記の構成設定は、ご使用の環境に合わせて変更できます。詳細については、「Apache HTTPD の設定」を参照してください。

Amazon Linux オペレーティングシステム間

Amazon Linux 2 および Amazon Linux 2023 オペレーティングシステム間の相違点の詳細については、「Amazon Linux 2023 ユーザーガイド」の「Amazon Linux 2 と Amazon Linux 2023 の比較」を参照してください。

Amazon Linux 2023 の詳細については、「Amazon Linux 2023 ユーザーガイド」の「Amazon Linux 2023 とは」を参照してください。

一般的な移行プロセス

本番稼働用に移行する準備ができたら、Elastic Beanstalk では、アップグレードを実行するために Blue/Green デプロイが必要です。ブルー/グリーンデプロイ手順での移行に推奨する一般的なベストプラクティス手順は次のとおりです。

移行テストの準備

アプリケーションをデプロイしてテストを開始する前に、前のセクション「相違点と互換性」の情報を確認してください。そのセクションに記載されているリファレンス、「Amazon Linux 2023 ユーザーガイド」の「Amazon Linux 2 と Amazon Linux 2023 の比較」も確認してください。このコンテンツのうち、ご使用のアプリケーションと設定のセットアップに当てはまる、または当てはまる可能性のある特定の情報を書き留めておいてください。

高レベル移行ステップ

1. AL2023 プラットフォームブランチをベースにした新しい環境を作成します。
2. ターゲットの AL2023 環境にアプリケーションをデプロイします。

   新しい環境のテストと調整を繰り返し行っている間、既存の本番環境は引き続きアクティブで影響を受けません。
3. 新しい環境でアプリケーションを徹底的にテストします。

4. ターゲットの AL2023 環境を本番環境に移行させる準備が完了したら、2 つの環境の CNAME を入れ替えて、新しい AL2023 環境にトラフィックをリダイレクトします。

より詳細な移行手順とベストプラクティス

ブルー/グリーンデプロイ手順の詳細については、「Elastic Beanstalk を使用したブルー/グリーンデプロイ」を参照してください。

より具体的なガイダンスと詳細なベストプラクティス手順については、「Blue/Green method」を参照してください。

移行計画に役立つその他のリファレンス

以下のリファレンスには、移行を計画するための追加情報があります。

- 「AWS Elastic Beanstalk プラットフォーム」の「Elastic Beanstalk でサポートされているプラットフォーム」
- リタイアしたプラットフォームブランチの履歴
- the section called “Linux プラットフォーム”
- プラットフォームの廃止に関するよくある質問

Amazon Linux AMI (AL1) から AL2 または AL2023 への移行

Elastic Beanstalk アプリケーションが Amazon Linux AMI プラットフォームブランチに基づいている場合について、このセクションでアプリケーションの環境を Amazon Linux 2 または Amazon Linux 2023 に移行する方法を説明します。Amazon Linux AMI に基づく以前の世代のプラットフォームブランチは、現在、廃止されています。

Amazon Linux 2023 は Amazon Linux 2 よりも新しいので、Amazon Linux 2023 に移行することを強くお勧めします。Amazon Linux 2 オペレーティングシステムは Amazon Linux 2023 よりも前にサポート終了となるため、Amazon Linux 2023 に移行すれば、より長いサポート期間を利用できるというメリットがあります。

着目すべきは、Elastic Beanstalk Amazon Linux 2 プラットフォームと Amazon Linux 2023 プラットフォームの間には高度な互換性があるという点です。ただし、インスタンスメタデータサービスバージョン 1 (IMDSv1) オプションのデフォルト、pkg-repo インスタンスツールのサポート、いくつかの Apache HTTPd 設定など、一部に違いがあります。詳細については、Amazon Linux 2023を参照してください。

相違点と互換性

AL2023/AL2 ベースのプラットフォームブランチは、既存のアプリケーションに対して下位互換性があるとは限りません。アプリケーションコードが新しいプラットフォームバージョンに正常にデプロイされた場合でも、オペレーティングシステムとランタイムの違いにより、動作やパフォーマンスが異なる場合があるという点を認識しておくことも重要です。

Amazon Linux AMI と AL2023/AL2 は同じ Linux カーネルを共有しますが、初期化システム、`libc` バージョン、コンパイラツールチェーン、およびさまざまなパッケージが異なります。詳細については、「Amazon Linux 2 に関するよくある質問」を参照してください。

Elastic Beanstalk サービスでは、プラットフォーム固有のバージョンのランタイム、ビルドツール、およびその他の依存関係も更新されました。

したがって、時間をかけて開発環境でアプリケーションを徹底的にテストし、必要な調整を行うことをお勧めします。

一般的な移行プロセス

本番稼働用に移行する準備ができたら、Elastic Beanstalk では、アップグレードを実行するために Blue/Green デプロイが必要です。ブルー/グリーンデプロイ手順での移行に推奨する一般的なベストプラクティス手順は次のとおりです。

移行テストの準備

アプリケーションをデプロイしてテストを開始する前に、このトピックの後に記載されている「すべての Linux プラットフォームに関する考慮事項」の情報を確認してください。また、それに続く「プラットフォーム固有の考慮事項」セクションで、ご使用のプラットフォームに該当する情報を確認してください。このコンテンツのうち、ご使用のアプリケーションと設定のセットアップに当てはまる、または当てはまる可能性のある特定の情報を書き留めておいてください。

高レベル移行ステップ

1. AL2 または AL2023 プラットフォームブランチをベースにした新しい環境を作成します。AL2023 プラットフォームブランチに移行することをお勧めします。
2. ターゲットの AL2023/AL2 環境にアプリケーションをデプロイします。

   新しい環境のテストと調整を繰り返し行っている間、既存の本番環境は引き続きアクティブで影響を受けません。
3. 新しい環境でアプリケーションを徹底的にテストします。

4. ターゲットの AL2023/AL2 環境を本番環境に移行させる準備が完了したら、2 つの環境の CNAME を入れ替えて、新しい環境にトラフィックをリダイレクトします。

より詳細な移行手順とベストプラクティス

ブルー/グリーンデプロイ手順の詳細については、「Elastic Beanstalk を使用したブルー/グリーンデプロイ」を参照してください。

より具体的なガイダンスと詳細なベストプラクティス手順については、「Blue/Green method」を参照してください。

移行計画に役立つその他のリファレンス

以下のリファレンスには、移行を計画するための追加情報があります。

- 「Amazon Linux 2023 ユーザーガイド」の「Amazon Linux 2 と Amazon Linux 2023 の比較」
- 「Amazon Linux 2023 ユーザーガイド」の「Amazon Linux 2023 とは」
- 「AWS Elastic Beanstalk プラットフォーム」の「Elastic Beanstalk でサポートされているプラットフォーム」
- リタイアしたプラットフォームブランチの履歴
- the section called “Linux プラットフォーム”
- プラットフォームの廃止に関するよくある質問

すべての Linux プラットフォームに関する考慮事項

次の表に、AL2023/AL2 へのアプリケーションの移行を計画する際に注意すべき考慮事項を示します。これらの考慮事項は、特定のプログラミング言語やアプリケーションサーバーに関係なく、任意の Elastic Beanstalk Linux プラットフォームに適用されます。

[面積]	変更と情報
設定ファイル	AL2023/AL2 プラットフォームでは、以前と同じように設定ファイルを使用でき、すべてのセクションは同じように動作します。ただし、特定の設定は、以前の Amazon Linux AMI プラットフォームと同じように機能しない場合があります。例: • 設定ファイルを使用してインストールしたソフトウェアパッケージの中には、AL2023/AL2 で使用できないものや、名前が変更されているものがあります。

[面積]	変更と情報
	• 一部のプラットフォーム固有の設定オプションは、プラットフォーム固有の名前空間から、プラットフォームに依存しない異なる名前空間に移行しました。 • `.ebextensions/nginx` ディレクトリで提供されるプロキシ設定ファイルは、`.platform/nginx` プラットフォームのフックディレクトリに移動する必要があります。詳細については、「リバースプロキシの設定」を参照してください。 環境インスタンスでカスタムコードを実行するには、プラットフォームフックを使用することをお勧めします。`.ebextensions` 設定ファイルでコマンドやコンテナコマンドを使用することはできますが、操作は簡単ではありません。たとえば、YAML ファイル内にコマンドスクリプトを記述することは、面倒でテストが難しい場合があります。 AWS CloudFormation リソースへのリファレンスが必要なスクリプトについては、`.ebextensions` 設定ファイルを使用する必要があります。
プラットフォームフック	AL2 プラットフォームは、実行可能ファイルを追加して環境のインスタンス上のディレクトリをフックすることで、環境のプラットフォームを拡張する新しい方法を導入します。以前の Linux プラットフォームバージョンでは、カスタムプラットフォームフックを使用していた可能性があります。これらのフックは、マネージドプラットフォーム用に設計されておらず、サポートされていませんでしたが、場合によっては便利な方法で動作する可能性があります。AL2023/AL2 プラットフォームバージョンでは、カスタムプラットフォームフックは機能しません。すべてのフックを新しいプラットフォームフックに移行する必要があります。詳細については、「プラットフォームフック」を参照してください。

[面積]	変更と情報
サポートされているプロキシサーバー	AL2023/AL2 プラットフォームのバージョンは、Amazon Linux AMI プラットフォームバージョンでサポートされている各プラットフォームと同じリバースプロキシサーバーをサポートします。ECS および Docker プラットフォームを除いて、AL2023/AL2 プラットフォームのすべてのバージョンは、デフォルトのリバースプロキシサーバーとして nginx を使用しています。Tomcat、Node.js、PHP、Python のプラットフォームも、代替として Apache HTTPD をサポートします。このセクションの説明の通り、すべてのプラットフォームでプロキシサーバーの設定が一貫して有効にされています。ただし、プロキシサーバーの設定は、Amazon Linux AMI の設定とは少し異なります。すべてのプラットフォームの相違点は次のとおりです。 • デフォルトは nginx - すべての AL2023/AL2 プラットフォームバージョンで、デフォルトのプロキシサーバーは nginx です。Tomcat、PHP、Python の Amazon Linux AMI プラットフォームバージョンでは、デフォルトのプロキシサーバーは Apache HTTPD でした。 • 一貫した名前空間 - すべての AL2023/AL2 プラットフォームバージョンでは、`aws:elasticbeanstalk:environment:proxy` 名前空間を使用してプロキシサーバーを設定します。Amazon Linux AMI プラットフォームのバージョンでは、プラットフォームごとの決定となり、Node.js は異なる名前空間を使用していました。 • 設定ファイルの場所 - プロキシ設定ファイルは、すべての AL2023/AL2 プラットフォームバージョンの `.platform/nginx` および `.platform/httpd` ディレクトリに配置する必要があります。Amazon Linux AMI プラットフォームのバージョンでは、これらの場所はそれぞれ `.ebextensions/nginx` と `.ebextensions/httpd` でした。 プラットフォーム固有のプロキシ設定の変更については、the section called “プラットフォーム固有の考慮事項” を参照してください。AL2023/AL2 プラットフォームでのプロキシ設定については、「リバースプロキシの設定」を参照してください。

[面積]	変更と情報
プロキシ設定の変更	各プラットフォームに固有のプロキシ設定の変更に加えて、すべてのプラットフォームに共通に適用されるプロキシ設定の変更があります。環境を正確に設定するには、両方を参照することが重要です。 • すべてのプラットフォーム – 「リバースプロキシの設定」を参照してください • プラットフォーム固有 – 「the section called “プラットフォーム固有の考慮事項”」を参照してください。
インスタンスプロファイル	AL2023/AL2 プラットフォームでは、インスタンスプロファイルを設定する必要があります。インスタンスプロファイルがなくても、環境の作成が一時的に成功する可能性はありますが、インスタンスプロファイルを必要とするアクションが失敗し始めると、作成直後に環境内にエラーが表示されることがあります。詳細については、「the section called “インスタンスプロファイル”」を参照してください。
拡張ヘルス	AL2023/AL2 プラットフォームのバージョンでは、デフォルトで拡張ヘルスが有効になります。これは、環境の作成に Elastic Beanstalk コンソールを使用しない場合、変更されます。コンソールでは、プラットフォームのバージョンに関係なく、可能な限り、デフォルトで拡張ヘルスが有効になります。詳細については、「the section called “拡張ヘルスレポートおよびモニタリング”」を参照してください。
カスタム AMI	環境でカスタム AMI を使用する場合は、Elastic Beanstalk AL2023/AL2 プラットフォームを使用して、新しい環境用に AL2023/AL2 に基づく新しい AMI を作成します。
カスタムプラットフォーム	AL2023/AL2 プラットフォームバージョンのマネージド AMI は、カスタムプラットフォームをサポートしていません。

プラットフォーム固有の考慮事項

このセクションでは、特定の Elastic Beanstalk Linux プラットフォームに固有の移行に関する考慮事項について説明します。

Docker

Amazon Linux AMI (AL1) に基づく Docker プラットフォームブランチファミリーには、3 つのプラットフォームブランチが含まれます。それぞれに個別の移行パスを設定することをお勧めします。

AL1 プラットフォームブランチ	AL2023/AL2 への移行パス
Amazon Linux AMI (AL1) 上で動作する Amazon ECS で管理されるマルチコンテナ Docker	ECS ベースの Docker AL2023/AL2 プラットフォームブランチ ECS ベースの Docker AL2023/AL2 プラットフォームブランチによって、マルチコンテナ Docker AL1 プラットフォームブランチで実行されている環境の移行パスを簡単にできます。 • 以前のマルチコンテナ Docker AL1 ブランチと同様に、AL2023/AL2 プラットフォームブランチは Amazon ECS を使用して、複数の Docker コンテナを Elastic Beanstalk 環境内の Amazon ECS クラスターにデプロイするように調整します。 • AL2023/AL2 プラットフォームブランチでは、以前のマルチコンテナ Docker AL1 ブランチのすべての機能がサポートされています。 • AL2023/AL2 プラットフォームブランチも同じ `Dockerrun.aws.json v2` ファイルをサポートしています。 マルチコンテナ Docker Amazon Linux プラットフォームブランチで動作するアプリケーションを AL2023/AL2 で動作する Amazon ECS プラットフォームブランチに移行する方法についての詳細は、「???」を参照してください。
Amazon Linux AMI (AL1) 上で動作する Docker Amazon Linux AMI (AL1) 上で動作する事前設定済み Docker (Glassfish 5.0)	AL2023/AL2 プラットフォームブランチで実行されている Docker 事前設定済み Docker (Glassfish 5.0) または Amazon Linux AMI (AL1) 上で動作する Docker に基づく環境で動作するアプリケーションは、Amazon Linux 2 で動作する Docker または AL2023 で動作する Docker のプラットフォームブランチに基づく環境に移行することをお勧めします。 ご使用の環境が、事前設定済み Docker (Glassfish 5.0) プラットフォームブランチ基づく場合は、「the section called “チュートリアル - Docker での GlassFish: Amazon Linux 2023 へのパス”」を参照してください。 次の表に、AL2023/AL2 で動作する Docker のプラットフォームブランチ固有の移行情報を示します。

AL1 プラットフォームブランチ	AL2023/AL2 への移行パス

[面積]	変更と情報
ストレージ	Elastic Beanstalk は、ストレージドライバを使用して Docker イメージとコンテナデータを保存するように Docker を設定します。Amazon Linux AMI では、Elastic Beanstalk はデバイスマッパーストレージドライバを使用しました。パフォーマンスを向上させるために、Elastic Beanstalk は追加の Amazon EBS ボリュームをプロビジョンしました。AL2023/AL2 Docker プラットフォームバージョンでは、Elastic Beanstalk は OverlayFS ストレージドライバを使用し、別のボリュームを必要とせずにパフォーマンスを向上させます。 Amazon Linux AMI では、`BlockDeviceMappings` 名前空間の `aws:autoscaling:launchconfiguration` オプションを使用して Docker 環境にカスタムストレージボリュームを追加した場合は、Elastic Beanstalk がプロビジョンする `/dev/xvdcz` Amazon EBS ボリュームも追加することをお勧めします。その後このボリュームを Elastic Beanstalk がプロビジョンすることはないので、設定ファイルから削除する必要があります。詳細については、「the section called “(Amazon Linux 2 より前の) Amazon Linux AMI での Docker 設定”」を参照してください。
プライベートリポジトリの認証	Docker で生成された認証ファイルを提供してプライベートリポジトリに接続するときに、Amazon Linux AMI Docker プラットフォームバージョンに必要な古い形式に変換する必要がなくなりました。AL2023/AL2 Docker プラットフォームバージョンは新しい形式をサポートしています。詳細については、「the section called “イメージリポジトリを使用した認証”」を参照してください。
プロキシサーバー	AL2023/AL2 Docker プラットフォームバージョンは、プロキシサーバーの背後で実行されないスタンドアロンコンテナをサポートしていません。Amazon Linux AMI Docker プラットフォームバージョンでは、以前は `aws:elasticbeanstalk:environment:pro`

AL1 プラットフォームブランチ	AL2023/AL2 への移行パス	
	[面積]	変更と情報
		xy 名前空間の ProxyServer オプションとして none の値を使用することができました。

Go

以下の表に、Go プラットフォームの AL2023/AL2 プラットフォームバージョンの移行情報を示します。

[面積]	変更と情報
ポートの受け渡し	AL2023/AL2 プラットフォームでは、Elastic Beanstalk は PORT 環境変数を使用してアプリケーションプロセスにポート値を渡しません。PORT 環境プロパティを自分で設定することで、プロセスのこの動作をシミュレートできます。ただし、複数のプロセスがあり、Elastic Beanstalk が増分ポート値をプロセス (5000、5100、5200 など) に渡すことを期待している場合は、実装を変更する必要があります。詳細については、「リバースプロキシの設定」を参照してください。

Amazon Corretto

次の表に、Java SE プラットフォームの Corretto プラットフォームブランチの移行情報を示します。

[面積]	変更と情報
Corretto と OpenJDK	Java プラットフォーム、Standard Edition (Java SE) を実装するために、AL2023/AL2 プラットフォームブランチは Open Java Development Kit (OpenJDK) の AWS ディストリビューションである Amazon Corretto を使用します。以前の Elastic Beanstalk Java SE プラットフォームブランチでは、Amazon Linux AMI に含まれている OpenJDK パッケージを使用しています。

[面積]	変更と情報
ビルドツール	AL2023/AL2 プラットフォームには、`gradle`、`maven`、および `ant` の新しいバージョンのビルドツールがあります。
JAR ファイルの処理	AL2023/AL2 プラットフォームでは、ソースバンドル (ZIP ファイル) に 1 つの JAR ファイルが含まれていて、他のファイルが含まれていない場合、Elastic Beanstalk は JAR ファイルの名前を `application.jar` に変更しなくなりました。名前変更が行われるのは、ZIP ファイル内ではなく、単独で JAR ファイルを送信した場合のみです。
ポートの受け渡し	AL2023/AL2 プラットフォームでは、Elastic Beanstalk は `PORT` 環境変数を使用してアプリケーションプロセスにポート値を渡しません。`PORT` 環境プロパティを自分で設定することで、プロセスのこの動作をシミュレートできます。ただし、複数のプロセスがあり、Elastic Beanstalk が増分ポート値をプロセス (5000、5100、5200 など) に渡すことを期待している場合は、実装を変更する必要があります。詳細については、「リバースプロキシの設定」を参照してください。
Java 7	Elastic Beanstalk は、AL2023/AL2 Java 7 プラットフォームブランチをサポートしていません。Java 7 アプリケーションをお持ちの場合は、Corretto 8 または Corretto 11 に移行してください。

Tomcat

以下の表に、Tomcat プラットフォームの AL2023/AL2 プラットフォームバージョンの移行情報を示します。

[面積]	変更と情報
設定オプション	AL2023/AL2 プラットフォームバージョンでは、Elastic Beanstalk は `aws:elasticbeanstalk:environment:proxy` 名前空間内の設定オプションとオプション値のサブセットのみをサポートします。各オプションの移行情報は次のとおりです。

[面積]	変更と情報
	<table><tr><th>オプション</th><th>移行情報</th></tr><tr><td>GzipCompression</td><td>AL2023/AL2 プラットフォームバージョンではサポートされていません。</td></tr><tr><td>ProxyServer</td><td>AL2023/AL2 Tomcat プラットフォームのバージョンは、nginx と Apache の HTTPD バージョン 2.4 プロキシサーバーの両方をサポートしています。しかし、Apache バージョン 2.2 はサポートされていません。 Amazon Linux AMI プラットフォームのバージョンでは、デフォルトのプロキシは Apache 2.4 でした。デフォルトのプロキシ設定を使用し、カスタムプロキシ設定ファイルを追加した場合、プロキシ設定は AL2023/AL2 で引き続き動作します。ただし、apache/2.2 オプション値を使用した場合は、プロキシ設定を Apache バージョン 2.4 に移行する必要があります。</td></tr></table> aws:elasticbeanstalk:container:tomcat:jvmoptions 名前空間の XX:MaxPermSize オプションは、AL2023/AL2 プラットフォームバージョンではサポートされていません。永続世代のサイズを変更するための JVM 設定は、Java 7 以前のバージョンにのみ適用されるため、AL2023/AL2 プラットフォームバージョンには適用されません。
アプリケーションのパス	AL2023/AL2 プラットフォームでは、環境の Amazon EC2 インスタンス上のアプリケーションのディレクトリへのパスは /var/app/current です。Amazon Linux AMI プラットフォームでは /var/lib/tomcat8/webapps でした。

Node.js

以下の表に、Node.js プラットフォームの AL2023/AL2 プラットフォームバージョンの移行情報を示します。

[面積]	変更と情報
インストールされている Node.js のバージョン	AL2023/AL2 プラットフォームでは、Elastic Beanstalk は複数の Node.js プラットフォームブランチを維持し、各プラットフォームバージョンのプラットフォームブランチに対応する Node.js メジャーバージョンの最新バージョンのみをインストールします。たとえば、Node.js 12 プラットフォームブランチの各プラットフォームバージョンには、デフォルトで Node.js 12.x.y のみがインストールされています。Amazon Linux AMI プラットフォームバージョンでは、各プラットフォームバージョンに複数の Node.js バージョンの複数のバージョンをインストールし、単一のプラットフォームブランチのみを維持しました。 アプリケーションで必要な Node.js メジャーバージョンに対応する Node.js プラットフォームブランチを選択します。
Apache HTTPD ログファイル名	AL2023/AL2 プラットフォームでは、Apache HTTPD プロキシサーバーを使用する場合、HTTPD ログファイル名は `access_log` および `error_log` になります。これは、Apache HTTPD をサポートする他のすべてのプラットフォームと一致します。Amazon Linux AMI プラットフォームのバージョンでは、これらのログファイルはそれぞれ `access.log` と `error.log` という名前でした。 すべてのプラットフォームのログファイル名と場所の詳細については、the section called “Elastic Beanstalk が CloudWatch Logs を設定する方法” を参照してください。
設定オプション	AL2023/AL2 プラットフォームでは、Elastic Beanstalk は `aws:elasticbeanstalk:container:nodejs` 名前空間の設定オプションをサポートしていません。オプションの中には、代替方法があります。各オプションの移行情報は次のとおりです。 | オプション	移行情報
`NodeCommand`	`package.json` ファイル内で `Procfile` または `scripts` キーワードを使用して、スタートスクリプトを指定します。
`NodeVersion`	`package.json` ファイル内で `engines` キーワードを使用して、Node.js のバージョンを指定します。指定できるのは、プラットフォームブランチに対応する Node.js バージョンのみであること

[面積]	変更と情報

オプション	移行情報
	に注意してください。たとえば、Node.js 12 プラットフォームブランチを使用している場合は、12.x.y Node.js バージョンのみを指定できます。詳細については、「the section called “package.json ファイルを使用した Node.js の依存関係の指定”」を参照してください。
`GzipCompression`	AL2023/AL2 プラットフォームバージョンではサポートされていません。

`ProxyServer`	AL2023/AL2 Node.js プラットフォームバージョンでは、このオプションは `aws:elasticbeanstalk:environment:proxy` 名前空間に移動しました。`nginx` (デフォルト) と `apache` のいずれかを選択できます。
	
	AL2023/AL2 Node.js プラットフォームバージョンは、プロキシサーバーの背後で実行されないスタンドアロンアプリケーションをサポートしていません。Amazon Linux AMI Node.js プラットフォームバージョンでは、以前は `aws:elasticbeanstalk:container:nodejs` 名前空間の `ProxyServer` オプションとして `none` の値を使用することができました。環境がスタンドアロンアプリケーションを実行している場合は、プロキシサーバー (nginx または Apache) がトラフィックを転送するポートをリッスンするようにコードを更新します。
	
	`var port = process.env.PORT || 5000;`
	
	`app.listen(port, function() {`
	`  console.log('Server running at http://127.0.0.1:%s',`
	` port);`
	`});`

PHP

以下の表に、PHP プラットフォームの AL2023/AL2 プラットフォームバージョンの移行情報を示します。

[面積]	変更と情報
PHP ファイル処理	AL2023/AL2 プラットフォームでは、PHP ファイルは PHP-FPM (CGI プロセスマネージャ) を使用して処理されます。Amazon Linux AMIプラットフォームでは、mod_php (Apache モジュール) を使用しました。
プロキシサーバー	AL2023/AL2 PHP プラットフォームのバージョンは、nginx と Apache HTTPD プロキシサーバーの両方をサポートします。デフォルトは nginx です。 Amazon Linux AMI PHP プラットフォームのバージョンは、Apache HTTPD のみをサポートします。カスタム Apache 設定ファイルを追加した場合は、`aws:elasticbeanstalk:environment:proxy` 名前空間の `ProxyServer` オプションを `apache` に設定できます。

Python

以下の表に、Python プラットフォームの AL2023/AL2 プラットフォームバージョンの移行情報を示します。

[面積]	変更と情報
WSGI サーバー	AL2023/AL2 プラットフォームでは、Gunicorn がデフォルトの WSGI サーバーです。デフォルトでは、Gunicorn はポート 8000 でリッスンします。このポートは、アプリケーションが Amazon Linux AMI プラットフォームで使用したものとは異なる場合があります。`aws:elasticbeanstalk:container:python` 名前空間の `WSGIPath` オプションを設定する場合は、値を Gunicorn の構文に置き換えます。詳細については、「the section called “Python 設定の名前空間”」を参照してください。 または、`Procfile` を使用して WSGI サーバーを指定および設定することもできます。詳細については、「the section called “[Procfile]”」を参照してください。

[面積]	変更と情報
アプリケーションのパス	AL2023/AL2 プラットフォームでは、環境の Amazon EC2 インスタンス上のアプリケーションのディレクトリへのパスは `/var/app/current` です。Amazon Linux AMI プラットフォームでは `/opt/python/current/app` でした。
プロキシサーバー	AL2023/AL2 Python プラットフォームのバージョンは、nginx と Apache HTTPD プロキシサーバーの両方をサポートします。デフォルトは nginx です。 Amazon Linux AMI Python プラットフォームのバージョンは、Apache HTTPD のみをサポートします。カスタム Apache 設定ファイルを追加した場合は、`aws:elasticbeanstalk:environment:proxy` 名前空間の `ProxyServer` オプションを `apache` に設定できます。

Ruby

以下の表に、Ruby プラットフォームの AL2023/AL2 プラットフォームバージョンの移行情報を示します。

[面積]	変更と情報
インストールされている Ruby バージョン	AL2023/AL2 プラットフォームでは、Elastic Beanstalk は、各プラットフォームバージョンに、プラットフォームブランチに対応する単一の Ruby バージョンの最新バージョンのみをインストールします。たとえば、Ruby 2.6 プラットフォームブランチの各プラットフォームバージョンには Ruby 2.6.x のみインストールされています。Amazon Linux AMI プラットフォームのバージョンでは、複数の Ruby バージョンの最新バージョン (2.4.x、2.5.x、2.6.x など) がインストールされています。 使用しているプラットフォームブランチに対応していない Ruby バージョンをアプリケーションで使用している場合は、アプリケーションに適した Ruby バージョンのプラットフォームブランチに切り替えることをお勧めします。
アプリケーションサーバー	AL2023/AL2 プラットフォームでは、Elastic Beanstalk は、すべての Ruby プラットフォームバージョンに対して Puma アプリケーションサーバーのみをインストールします。`Procfile` を使用して別のアプリケーションサーバーをスタートし、`Gemfile` を使用してインストールできます。

[面積]	変更と情報
	Amazon Linux AMI プラットフォームでは、Ruby バージョンごとに 2 種類のプラットフォームブランチをサポートしていました。1 つは Puma アプリケーションサーバーで、もう 1 つは Passenger アプリケーションサーバーです。アプリケーションで Passenger を使用している場合は、Passenger をインストールして使用するように Ruby 環境を設定できます。 詳細な説明と例については the section called “Ruby プラットフォーム” を参照してください。

プラットフォームの廃止に関するよくある質問

> **Note**
>
> 2022 年 7 月 18 日に、Elastic Beanstalk では Amazon Linux AMI (AL1) に基づくプラットフォームブランチがすべて廃止されました。

この FAQ の回答は、次のトピックを参照しています。

- Elastic Beanstalk プラットフォームのサポートポリシー
- リタイアしたプラットフォームブランチの履歴
- 「AWS Elastic Beanstalk プラットフォーム」の「Elastic Beanstalk でサポートされているプラットフォーム」
- Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する
- Amazon Linux 2 に関するよくある質問。

1. プラットフォームブランチのリタイアとはどういう意味ですか。

発表されたプラットフォームブランチのリタイア日が過ぎると、そのプラットフォームブランチに基づくアクティブな環境が既にある場合を除き、リタイアしたプラットフォームブランチに基づいて新しい環境を作成できなくなります。詳細については、FAQ #11 を参照してください。Elastic Beanstalk では、これらのプラットフォームブランチに対する新しいメンテナンス更新プログラムの

提供が停止されます。リタイアしたプラットフォームブランチは、本番環境での使用にはお勧めしません。詳細については、FAQ #5 を参照してください。

2. が AL1-basedプラットフォームブランチを AWS 廃止したのはなぜですか？

Elastic Beanstalk では、ベンダーがプラットフォームコンポーネントを非推奨またはリタイアにしたときに、プラットフォームブランチがリタイアになります。この場合に該当して、2020 年 12 月 31 日現在、Amazon Linux AMI (AL1) の標準サポートは終了しています。Elastic Beanstalk は 2022 年まで AL1 ベースのプラットフォームを提供し続けましたが、その間、最新の機能を備えた AL2 および AL2023 ベースのプラットフォームがリリースされました。お客様が今後も最新のセキュリティと機能の恩恵を受けるためには、AL2 または AL2023 ベースのプラットフォームに移行することが重要です。

3. リタイアしたプラットフォームブランチはどれですか。

廃止されたプラットフォームコンポーネントとプラットフォームブランチのリストについては、リタイアしたプラットフォームブランチの履歴 を参照してください。

4. 現在サポートされているプラットフォームはどれですか。

「AWS Elastic Beanstalk プラットフォーム」の「Elastic Beanstalk でサポートされているプラットフォーム」を参照してください。

5. Elastic Beanstalk では、リタイア後に環境のコンポーネントは削除または終了されますか。

廃止されたプラットフォームブランチのポリシーでは、環境へのアクセスを削除したり、リソースを削除したりすることはありません。ただし、リタイアしたプラットフォームブランチに基づく環境は、予測不能な状況になることがあります。サプライヤーがコンポーネントを End of Life (EOL) とマークしているので、リタイアしたプラットフォームブランチに対して Elastic Beanstalk でセキュリティ更新プログラム、テクニカルサポート、修正プログラムを提供できないためです。例えば、リタイアしたプラットフォームブランチで実行されている環境に、有害で重大なセキュリティ脆弱性が現れる可能性があります。または、時間の経過とともに Elastic Beanstalk サービスとの互換性がなくなると、環境での EB API アクションの動作が停止することがあります。このようなリスクの可能性は、リタイアしたプラットフォームブランチに基づく環境が長くアクティブなままであるほど、増加します。

廃止されたプラットフォームブランチでの実行中にアプリケーションに問題が発生し、サポートされているプラットフォームに移行できない場合は、他の代替方法を検討する必要があります。回避策と

して、アプリケーションを Docker イメージにカプセル化して Docker コンテナとして実行する方法があります。これにより、お客様は、Elastic Beanstalk AL2023/AL2 Docker プラットフォームなどの当社の Docker ソリューションや、Amazon ECS や Amazon EKS などのその他の Docker ベースのサービスを使用できるようになります。Docker 以外の代替方法には、必要なランタイムを完全にカスタマイズできる AWS CodeDeploy サービスが含まれます。

6. リタイア日の延期を申請できますか。

いいえ。リタイア日以降、既存の環境は機能し続けます。ただし、Elastic Beanstalk ではプラットフォームのメンテナンスとセキュリティアップデートが提供されなくなります。そのため、AL1 ベースのプラットフォームでまだアプリケーションを実行している場合は、AL2 または AL2023 に移行することが重要です。リスクと回避策の詳細については、FAQ #5 を参照してください。

7. AL2 または AL2023 への移行を期間内に完了できない場合の回避策を教えてください。

お客様は引き続き環境を実行することもできますが、すべての Elastic Beanstalk 環境を、サポートされているプラットフォームバージョンに移行するように計画することを強くお勧めします。移行することにより、リスクが最小限に抑えられ、最近のリリースで提供される重要なセキュリティ、パフォーマンス、および機能拡張のメリットを継続的に利用できるようになります。リスクと回避策の詳細については、FAQ #5 を参照してください。

8. AL2 または AL2023 プラットフォームへの移行に推奨されるプロセスは何ですか。

AL1 から AL2023/AL2 への包括的な移行手順については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。このトピックでは、Elastic Beanstalk でアップグレードを実行するためにブルー/グリーンデプロイが必要であることを説明します。

9. リタイアしたプラットフォームで環境を実行している場合、どのような影響がありますか。

リタイアしたプラットフォームブランチに基づく環境は、予測不能な状況になることがあります。サプライヤーがコンポーネントを End of Life (EOL) とマークしているので、リタイアしたプラットフォームブランチに対して Elastic Beanstalk でセキュリティ更新プログラム、テクニカルサポート、修正プログラムを提供できないためです。例えば、リタイアしたプラットフォームブランチで実行されている環境に、有害で重大なセキュリティ脆弱性が現れる可能性があります。または、時間の経過とともに Elastic Beanstalk サービスとの互換性がなくなると、環境での EB API アクションの動

作が停止することがあります。このようなリスクの可能性は、リタイアしたプラットフォームブランチ上の環境が長くアクティブなままであるほど、増加します。詳細については、FAQ #5 を参照してください。

10. リタイア日の 90 日後にどうなりますか。

廃止されたプラットフォームブランチのポリシーでは、環境へのアクセスを削除したり、リソースを削除したりすることはありません。ただし、リタイアしたプラットフォームブランチに基づく環境は、予測不能な状況になることに注意してください。サプライヤーがコンポーネントを End of Life (EOL) とマークしているので、リタイアしたプラットフォームブランチに対して Elastic Beanstalk でセキュリティ更新プログラム、テクニカルサポート、修正プログラムを提供できないためです。例えば、リタイアしたプラットフォームブランチで実行されている環境に、有害で重大なセキュリティ脆弱性が現れる可能性があります。または、時間の経過とともに Elastic Beanstalk サービスとの互換性がなくなると、環境での EB API アクションの動作が停止することがあります。このようなリスクの可能性は、リタイアしたプラットフォームブランチ上の環境が長くアクティブなままであるほど、増加します。詳細については、FAQ #5 を参照してください。

11. リタイアしたプラットフォームに基づいて、新しい環境を作成できますか。

既にそのプラットフォームブランチを使用して、同じアカウントで同じリージョンに既存の環境を作成している場合は、リタイアしたプラットフォームブランチに基づいて新しい環境を作成できます。廃止されたプラットフォームブランチは Elastic Beanstalk コンソールでは使用できません。ただし、リタイアしたプラットフォームブランチに基づく既存の環境があるお客様は、EB CLI、EB API、および AWS CLIを通じて使用できます。また、既存のお客様は [Clone environment] (環境のクローン作成) コンソールと [Rebuild environment] (環境の再構築) コンソールを使用できます。ただし、リタイアしたプラットフォームブランチに基づく環境は、予期しない状況に陥る可能性があることに注意してください。詳細については、FAQ #5 を参照してください。

12. 廃止されたプラットフォームブランチで既存の環境を実行している場合、廃止されたプラットフォームブランチに基づいて新しい環境を作成できるのはいつまでですか? コンソール、CLI、または API を使用して作成できますか。

環境は、廃止日以降に作成できます。ただし、リタイアしたプラットフォームブランチは、予測不可能な状況に陥る可能性があることに注意してください。このような環境が作成されたりアクティブになったりする時期が遅くなるほど、環境に予期せぬ問題が発生するリスクが高くなります。新しい環境の作成についての詳細は、FAQ #11 を参照してください。

13. リタイアしたプラットフォームに基づく環境のクローン作成または再構築はできますか。

はい。これを行うには、[Clone environment] (環境のクローン作成) コンソールと [Rebuild environment] (環境の再構築) コンソールを使用します。EB CLI、EB API、および を使用することもできます AWS CLI。新しい環境の作成についての詳細は、FAQ #11 を参照してください。

すべての Elastic Beanstalk 環境を、サポートされているプラットフォームバージョンに移行するように計画することを強くお勧めします。移行することにより、リスクが最小限に抑えられ、最近のリリースで提供される重要なセキュリティ、パフォーマンス、および機能拡張のメリットを継続的に利用できるようになります。リスクと回避策の詳細については、FAQ #5 を参照してください。

14. リタイア日以降、リタイアしたプラットフォームブランチに基づく Elastic Beanstalk 環境の AWS リソースはどうなりますか？ 例えば、実行中の EC2 インスタンスが終了した場合、Elastic Beanstalk は容量を維持するために新しい AL1 ベースの EC2 インスタンスを起動できますか。

環境のリソースはアクティブなままであり、機能し続けます。はい、Elastic Beanstalk は環境内の AL1 EC2 インスタンスの自動スケーリングを行います。ただし、Elastic Beanstalk では、環境への新しいプラットフォームメンテナンス更新プログラムの提供が停止され、環境が時間の経過とともに予測不可能な状況に陥る可能性があります。詳細については、FAQ #5 を参照してください。

15. AL2023/AL2 と Amazon Linux AMI (AL1) のオペレーティングシステムの主な違いは何ですか。Elastic Beanstalk AL2023/AL2 プラットフォームブランチにどのような影響がありますか。

Amazon Linux AMI と AL2023/AL2 は同じ Linux カーネルを共有しますが、初期化システム、`libc` バージョン、コンパイラツールチェーン、およびさまざまなパッケージが異なります。詳細については、「Amazon Linux 2 に関するよくある質問」を参照してください。

Elastic Beanstalk サービスでは、プラットフォーム固有のバージョンのランタイム、ビルドツール、およびその他の依存関係も更新されました。AL2023/AL2 ベースのプラットフォームブランチは、既存のアプリケーションに対して下位互換性があるとは限りません。さらに、アプリケーションコードが新しいプラットフォームバージョンに正常にデプロイされた場合でも、オペレーティングシステムとランタイムの違いにより、動作やパフォーマンスが異なる場合があります。確認およびテストする必要のある設定とカスタマイズのリストと説明については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

環境設定の更新およびアプリケーションのデプロイのキャンセル

環境設定の変更によってトリガされる進行中の更新はキャンセルできます。また、進行中の新しいアプリケーションバージョンのデプロイをキャンセルすることもできます。たとえば、新しい環境設定を適用するのではなく、既存の環境設定を引き続き使用するよう決定した場合は、更新をキャンセルすることもあります。または、デプロイする新しいアプリケーションバージョンに問題があり、それによってアプリケーションが起動しなくなったり、正しく実行されなくなったりすることがわかる場合があります。環境またはアプリケーションバージョンの更新をキャンセルすることで、更新またはデプロイプロセスが終了するのを待ってから、環境またはアプリケーションバージョンの更新を新たに試みることを回避できます。

> **Note**
>
> 不要になった古いリソースが削除されるクリーンアップフェーズでは、インスタンスの最後のバッチが更新された後に、更新をキャンセルすることはできません。

Elastic Beanstalk は、最後に成功した更新と同じ方法を使ってロールバックを実行します。例えば、環境で時間ベースのローリング更新を有効にしている場合、Elastic Beanstalk はインスタンスの 1 つのバッチでの変更のロールバックから、次のバッチの変更のロールバックまで、指定された時間待機します。または、最近ローリング更新を有効にしたが、最後に正常に環境設定を更新した際に、ローリング更新を行わなかった場合、Elastic Beanstalk ではすべてのインスタンスで同時にロールバックが実行されます。

いったん更新のキャンセルを開始すると、Elastic Beanstalk で前の環境設定にロールバックすることはできません。ロールバックプロセスは、環境のすべてのインスタンスに以前の環境設定が戻るか、ロールバックプロセスが失敗するまで続行されます。アプリケーションバージョンのデプロイでは、デプロイをキャンセルすると単純にデプロイが中止されます。一部のインスタンスには新しいアプリケーションバージョンが適用され、他のインスタンスは引き続き既存のアプリケーションバージョンが実行されます。後で、同じアプリケーションバージョンまたは別のアプリケーションバージョンをデプロイできます。

ローリング更新の詳細については、「Elastic Beanstalk 環境設定のローリング更新」を参照してください。バッチ処理されたアプリケーションバージョンのデプロイの詳細については、「デプロイポリシーと設定」を参照してください。

更新をキャンセルするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Actions] (アクション) を選択してから、[Abort current operation] (現在のオペレーションを中止) を選択します。

Elastic Beanstalk 環境の再構築

Elastic Beanstalk 機能を使用して AWS Elastic Beanstalk 環境の基盤となる AWS リソースを変更または終了しないと、環境が使用できなくなる可能性があります。その場合は、環境を再構築して、動作状態になるよう復元を試みることができます。環境を再構築すると、すべてのリソースが終了し、同じ設定の新しいリソースに置き換えられます。

また、6 週間 (42 日) 以内であれば、終了した環境を再構築できます。再構築の際、Elastic Beanstalk は同じ名前、ID、および設定で新しい環境の作成を試みます。

実行中の環境の再構築

Elastic Beanstalk コンソールまたは `RebuildEnvironment` API を使用して、環境を再構築できます。

> **Warning**
> 環境にデータベースが結合されている場合、再構築の一部として削除され、再構築された環境の新しいデータベースには以前のデータが含まれません。データベースを保持したり、スナップショットを作成したりする場合は、再構築後に目的の結果に合わせてデータベース削除ポリシーが正しく設定されていることを確認してください。詳細については、「データベースのライフサイクル」を参照してください。

実行中の環境 (コンソール) を再構築するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] を選択してから、[Rebuild environment] を選択します。
4. [Rebuild] を選択します。

Elastic Beanstalk API を使用して実行中の環境を再構築するには、 AWS CLI または AWS SDK で `RebuildEnvironment`アクションを使用します。

```
$ aws elasticbeanstalk rebuild-environment --environment-id e-vdnftxubwq
```

終了した環境の再構築

終了した環境を再構築し、復元するには、Elastic Beanstalk コンソール、EB CLI、または `RebuildEnvironment` API を使用できます。

> **Note**
>
> 終了済み環境でカスタムドメイン名を使用しない場合、環境は elasticbeanstalk.com のサブドメインを使用します。これらのサブドメインは、Elastic Beanstalk リージョン内で共有されます。したがって、同じリージョンで別のお客様が作成した別の環境にも使用できます。環境が終了済みであれば、別の環境でそのサブドメインを使用できます。この場合、再構築は失敗します。
> この問題は、カスタムドメインを使用することで回避できます。詳細については、「Elastic Beanstalk 環境のドメイン名」を参照してください。

最近終了した環境は、最大で 1 時間はアプリケーションの概要に表示されます。この間は、そのダッシュボードで環境のイベントを表示し、[Restore environment] アクションを使用してイベントを再構築できます。

表示されなくなった環境を再構築するには、アプリケーションページの [Restore terminated environment] オプションを使用します。

終了した環境を再構築するには (コンソール)

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. [Actions] を選択してから、[Restore terminated environment] を選択します。
4. 終了した環境を選択します。
5. [復元] を選択します。

Elastic Beanstalk は、同じ名前、ID、および設定で新しい環境の作成を試みます。再構築を試みるときに同じ名前または URL を持つ環境が存在する場合、再構築は失敗します。環境にデプロイされたアプリケーションバージョンを削除した場合も、再構築に失敗します。

EB CLI を使用して環境を管理する場合は、eb restore コマンドを使用して、終了した環境を再構築します。

```
$ eb restore e-vdnftxubwq
```

詳細については「eb restore」を参照してください。

Elastic Beanstalk API を使用して終了した環境を再構築するには、 AWS CLI または AWS SDK で RebuildEnvironment アクションを使用します。

```
$ aws elasticbeanstalk rebuild-environment --environment-id e-vdnftxubwq
```

環境タイプ

では AWS Elastic Beanstalk、負荷分散されたスケーラブルな環境または単一インスタンス環境を作成できます。必要な環境タイプは、デプロイしたアプリケーションによって異なります。たとえば、コストの節約のためシングルインスタンス環境でアプリケーションを開発およびテストし、アプリケーションが本番環境向けに整ってから、環境を負荷分散されたスケーラブルな環境にアップグレードできます。

> **Note**
>
> バックグラウンドのタスクを処理するウェブアプリケーションのワーカー環境枠には、ロードバランサーは含まれません。ただし、ロードによって必要とされるときに Amazon SQS キューからデータを処理できるように、Auto Scaling グループにインスタンスを追加することで、ワーカー環境が効果的にスケールアウトされます。

負荷分散されたスケーラブルな環境

負荷分散されたスケーラブルな環境では、Elastic Load Balancing サービスと Amazon EC2 Auto Scaling サービスを使用して、デプロイされたアプリケーションに必要な Amazon EC2 インスタンスをプロビジョニングします。Amazon EC2 Auto Scaling は、アプリケーションへの負荷の増大に

対応できるように追加インスタンスを自動的に開始します。アプリケーションへの負荷が軽減すると、Amazon EC2 Auto Scaling はインスタンスを停止しますが、最低インスタンス実行数は常に保持されます。アプリケーションで、複数のアベイラビリティーゾーンで実行するオプションを備えたスケーラビリティが必要な場合、負荷分散されたスケーラブルな環境を使用します。どの環境タイプを選択したらよいか分からない場合は、1 つ選び、必要に応じて後から環境タイプを変更することができます。

シングルインスタンス環境

シングルインスタンス環境には、Elastic IP アドレスを使用する 1 つの Amazon EC2 インスタンスが含まれています。シングルインスタンス環境にはロードバランサーがなく、負荷分散されたスケーラブルな環境と比べてコストを節約できます。単一インスタンス環境でも Amazon EC2 Auto Scaling サービスは使用されますが、インスタンスの最小数、インスタンスの最大数、および適切な容量の設定はすべて 1 に設定されています。したがって、アプリケーションで増加する負荷に対応するために新しいインスタンスが起動されるということはありません。

本番アプリケーションのトラフィックが少ないと考えられる場合や、リモート開発を行っている場合は、シングルインスタンス環境を使用します。どの環境タイプを選択したらよいか分からない場合は、1 つ選び、必要に応じて後から環境タイプを変更します。詳細については、「環境タイプの変更」を参照してください。

環境タイプの変更

環境タイプをシングルインスタンスまたは負荷分散されたスケーラブルな環境に変更するには、環境設定を編集します。場合によっては、環境タイプを別のタイプに変更することが必要になる場合があります。たとえば、コストの節約のため、シングルインスタンス環境でアプリケーションを開発しテストするとします。アプリケーションが本番環境で稼働できるようになったら、環境タイプを負荷分散されたスケーラブルな環境に変更して、お客様の需要に対応して拡張することができます。

環境のタイプを変更するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [キャパシティー] カテゴリで、[編集] を選択します。
5. [環境タイプ] リストで、目的の環境タイプを選択します。

6. [保存] を選択します。

   Elastic Beanstalk が AWS リソースをプロビジョニングしている間、環境が更新されるまでに数分かかることがあります。

環境が VPC 内にある場合は、Elastic Load Balancing と Amazon EC2 インスタンスを配置するサブネットを選択します。アプリケーションを実行する各アベイラビリティーゾーンでは両方が必要です。詳細については、「Amazon VPC で Elastic Beanstalk を使用する」を参照してください。

Elastic Beanstalk ワーカー環境

AWS Elastic Beanstalk アプリケーションが、完了までに時間がかかるオペレーションまたはワークフローを実行する場合は、それらのタスクを専用のワーカー環境にオフロードできます。ブロックオペレーションを実行するプロセスからウェブアプリケーションのフロントエンドを分離することは、アプリケーションに負荷がかかっていても応答性を保つための一般的な方法です。

長い時間実行されるタスクとは、イメージやビデオの処理、E メールの送信、ZIP アーカイブの生成など、リクエストの完了までの時間を実質的に増やすタスクです。これらのオペレーションの完了には 1~2 秒しかかからない場合がありますが、数秒の遅延であっても、それ以外の場合は 500 ms 以下で完了するウェブリクエストにとっては大きなものです。

1 つのオプションでは、ワーカープロセスをローカルにスポーンし、成功を返して、タスクを非同期的に処理します。インスタンスが、それに送信されたすべてのタスクに追い付くことができれば、この方法は有効です。ただし、高い負荷がかかった場合、インスタンスはバックグラウンドタスクを処理しきれなくなり、より優先度の高いリクエストに応答しなくなる可能性があります。個別のユーザーが複数のタスクを生成できる場合、負荷の増加はユーザーの増加に対応できず、ウェブサーバー層を効果的にスケールアウトすることが困難になる可能性があります。

長時間実行されるタスクをローカルで実行しないようにするには、プログラミング言語の AWS SDK を使用して Amazon Simple Queue Service (Amazon SQS) キューに送信し、別のインスタンスセットで実行するプロセスを実行できます。次に、ワーカーインスタンスにキャパシティーがある場合に限り、キューから項目を取り出して実行することで、処理能力を超えないようにします。

Elastic Beanstalk のワーカー環境では、このプロセスを簡素化するために Amazon SQS キューを管理し、キューから読み取る各インスタンスでデーモンプロセスを自動的に実行します。デーモンは、キューから項目を取得すると、キューメッセージの内容を本文に含めて、HTTP POST リクエストをローカルにポート 80 の `http://localhost/` に送信します。アプリケーションに必要なのは、POST に応じて実行時間が長いタスクを実行することだけです。デーモンを設定し、別のパスに

送信する、application/JSON 以外の MIME タイプを使用する、既存のキューに接続する、または接続 (最大の同時リクエスト数)、タイムアウト、再試行をカスタマイズすることができます。

定期的なタスクでは、cron スケジュールに基づいてメッセージをキューに入れるようワーカーデーモンを設定することもできます。定期的な各タスクでは、別のパスに POST できます。各タスクのスケジュールおよびパスを定義するソースコードに YAML ファイルを含めて、定期的なタスクを有効にします。

> **Note**
>
> Windows Server プラットフォームの .NET はワーカー環境をサポートしていません。

セクション

- ワーカー環境 SQS デーモン
- デッドレターキュー
- 定期的なタスク
- ワーカー環境枠での自動スケーリングのための Amazon CloudWatch の使用
- ワーカー環境の設定

ワーカー環境 SQS デーモン

ワーカー環境は、Elastic Beanstalk から提供されるプロセスデーモンを実行します。このデーモンは定期的に更新され、機能の追加とバグの修正が行われます。デーモンの最新バージョンを取得するには、最新のプラットフォームバージョンに更新します。

ワーカー環境内のアプリケーションが `200 OK` レスポンスを返し、リクエストを受信して正常に処理したことを確認すると、デーモンが `DeleteMessage` 呼び出しを Amazon SQS キューに送信し、そのメッセージをキューから削除します。アプリケーションが `200 OK` 以外の応答を返した場合、Elastic Beanstalk は、設定済みの `ErrorVisibilityTimeout` 期間の経過後にメッセージをキューに戻します。応答がない場合、Elastic Beanstalk は、処理中の別の試行でそのメッセージを使用できるように、`InactivityTimeout` 期間の経過後にメッセージをキューに戻します。

> **Note**
>
> Amazon SQSキューのプロパティ (メッセージの順序、少なくとも 1 回の配信、およびメッセージのサンプリング) は、ワーカー環境のウェブアプリケーションをどのように設計するかに影響する可能性があります。詳細については、Amazon Simple Queue Service デベロッパーガイドの「Properties of Distributed Queues (分散キューのプロパティ)」を参照してください。

Amazon SQS では、キューにあるメッセージが、設定した `RetentionPeriod` を超過すると自動的に削除されます。

デーモンは以下の HTTP ヘッダーを設定します。

HTTP ヘッダー

名前	値

`User-Agent`	`aws-sqsd`
	
	`aws-sqsd/1.1 1`

`X-Aws-Sqsd-Msgid`	メッセージストーム (異常に多数の新規メッセージなど) を検出するために使用される SQS メッセージ ID
`X-Aws-Sqsd-Queue`	SQS キューの名前。
`X-Aws-Sqsd-First-Received-At`	メッセージを最初に受信したときの ISO 8601 形式の UTC 時間。
`X-Aws-Sqsd-Receive-Count`	SQS メッセージの受信件数。
`X-Aws-Sqsd-Attr-` *`message-attribute-name`*	処理されるメッセージに割り当てられたカスタムメッセージ属性。`message-attribute-name` は実際のメッセージ属性名です。文字列と数値のメッセージ属性はすべてヘッダーに追加されますが、バイナリ属性は破棄され、ヘッダーに含まれません。

HTTP ヘッダー	
`Content-Type`	Mime タイプ設定、デフォルトでは `application/json` 。

デッドレターキュー

Elastic Beanstalk ワーカー環境では、Amazon Simple Queue Service (Amazon SQS) のデッドレターキューがサポートされています。デッドレターキューは、何らかの理由で正常に処理できなかったメッセージを他の（送信元）キューが送信できるキューです。デッドレターキューを使用することの主なメリットは、正常に処理されなかったメッセージを対象から外して隔離することができることです。その後、デッドレターキューに送信されたメッセージを分析し、正常に処理されなかった理由を調べることができます。

ワーカー環境の作成時に、自動生成された Amazon SQS キューを指定した場合、そのワーカー環境ではデッドレターキューがデフォルトで有効になっています。ワーカー環境に対して既存の SQS キューを選択した場合、SQS を使用してデッドレターキューを個別に設定する必要があります。SQS を使用してデッドレターキューを設定する方法については、「Amazon SQS デッドレターキューの使用」を参照してください。

デッドレターキューを無効にすることはできません。配信できないメッセージは、最終的には必ずデッドレターキューに送信されます。ただし、実質的にはこの機能を無効にできます。そのためには、`MaxRetries` オプションを有効な最大値 100 に設定します。

ワーカー環境の Amazon SQS キューにデッドレターキューが設定されていない場合、Amazon SQS は保持期間が終了するまでメッセージをキューに保持します。保存期間の設定の詳細については、「the section called “ワーカー環境の設定”」を参照してください。

> **Note**
>
> Elastic Beanstalk の `MaxRetries` オプションは、SQS の `MaxReceiveCount` オプションと同じです。ワーカー環境が自動生成された SQS キューを使用しない場合は、SQS で `MaxReceiveCount` オプションを使用してデッドレターキューを無効にできます。詳細については、「Amazon SQS デッドレターキューの使用」を参照してください。

SQS メッセージのライフサイクルの詳細については、「メッセージのライフサイクル」を参照してください。

定期的なタスク

ソースバンドルで `cron.yaml` という名前のファイルに定期的なタスクを定義し、定期的な間隔でワーカー環境のキューにジョブを自動的に追加できます。

たとえば、次の `cron.yaml` ファイルは 2 つの定期的なタスクを作成します。最初のタスクは 12 時間ごとに実行され、2 番目のタスクは毎日午後 11 時 (UTC) に実行されます。

Example cron.yaml

```
version: 1
cron:
 - name: "backup-job"
   url: "/backup"
   schedule: "0 */12 * * *"
 - name: "audit"
   url: "/audit"
   schedule: "0 23 * * *"
```

name は、各タスクに対して一意である必要があります。URL は、ジョブをトリガーするために POST リクエストを送信するパスです。スケジュールは、タスクをいつ実行するかを決定する CRON 式です。

タスクを実行すると、デーモンは実行する必要があるジョブを示すヘッダーとともに環境の SQS キューにメッセージをポストします。環境の任意のインスタンスはメッセージを取得し、ジョブを処理できます。

> **Note**
>
> 既存の SQS キューを使用し、Amazon SQS FIFO キューを選択してワーカー環境を構成している場合、定期的なタスクはサポートされません。

Elastic Beanstalk はリーダーの選択を使用して、ワーカー環境で定期的なタスクをキューに入れるインスタンスを決定します。各インスタンスは、Amazon DynamoDB テーブルに書き込むことで、リーダーになろうとします。成功する最初のインスタンスはリーダーであり、リーダーのステータスを維持するには、テーブルに書き込み続ける必要があります。リーダーがサービス停止状態となった場合、すぐに別のインスタンスに置き換えられます。

定期的なタスクでは、ワーカーデーモンは以下の追加のヘッダーを設定します。

HTTP ヘッダー

名前	値
`X-Aws-Sqsd-Taskname`	定期的なタスクでは、実行するタスクの名前。
`X-Aws-Sqsd-Scheduled-At`	定期的なタスクが予定されている時刻
`X-Aws-Sqsd-Sender-Id`	AWS メッセージの送信者の アカウント番号

ワーカー環境枠での自動スケーリングのための Amazon CloudWatch の使用

Amazon EC2 Auto Scaling と CloudWatch は連動して、ワーカー環境で実行中のインスタンスの CPU 使用率をモニタリングします。CPU 処理能力の自動スケーリング制限の設定により、Amazon SQS キューにあるメッセージのスループットを適切に管理するため Auto Scaling グループが実行するインスタンスの数が決定されます。それぞれの EC2 インスタンスから、その CPU の使用状況メトリクスが CloudWatch へ発行されます。Amazon EC2 Auto Scaling は、CloudWatch から、ワーカー環境内のすべてのインスタンスの平均 CPU 使用率を取得します。CPU 能力に応じて、追加または終了するインスタンスの数、および上限と下限のしきい値を設定します。Amazon EC2 Auto Scaling によって、指定した CPU 処理能力の上限しきい値に達したことが検出されると、Elastic Beanstalk がワーカー環境に新しいインスタンスを作成します。これらのインスタンスは、CPU 負荷がしきい値未満に戻ると削除されます。

> **Note**
>
> インスタンスの削除時に処理されていなかったメッセージは、キューに戻され、まだ実行中であるインスタンスの別のデーモンで処理されます。

また、必要に応じて Elastic Beanstalk コンソール、CLI、またはオプションのファイルを使用して、別の CloudWatch アラームを設定することもできます。詳細については、「「Amazon CloudWatch で Elastic Beanstalk を使用する」」および「ステップスケーリングポリシーを使用して Auto Scaling グループを作成する」を参照してください。

ワーカー環境の設定

ワーカー環境の設定は、環境マネジメントコンソールで、環境の [Configuration] (設定) ページの [Worker] (ワーカー) カテゴリを編集することで管理できます。

> **Note**
>
> ワーカーキューメッセージを送信する URL パスを設定できますが、IP ポートを設定することはできません。Elastic Beanstalk は、常にワーカーキューメッセージをポート 80 に送信します。ワーカー環境アプリケーションまたはそのプロキシは、ポート 80 をリッスンする必要があります。

ワーカーデーモンを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [ワーカー] 設定カテゴリで、[編集] を選択します。

[ワーカー変更] 設定ページには、次のオプションがあります。

[キュー] セクションで、次の操作を行います。

- [ワーカーキュー] – デーモンにより読み込まれる Amazon SQS キューを指定します。既存のキューがあればそれを選択することができます。[自動生成されたキュー] を選択すると、Elastic Beanstalk によって新しい Amazon SQS キューおよび対応する [ワーカーキュー URL] が作成されます。

> **Note**
>
> [自動生成されたキュー] を選択した場合、Elastic Beanstalk によって作成されるキューは標準の Amazon SQS キューです。既存のキューを選択する場合は、標準キューまたは FIFO Amazon SQS キューのいずれかを指定できます。FIFO キューを指定する場合、定期的なタスクはサポートされないことに注意してください。

- [ワーカーキュー URL] – 既存の [ワーカーキュー] を選択すると、その Amazon SQS キューに関連付けられている URL が表示されます。

[メッセージ] セクションで、次の操作を行います。

- [HTTP パス] – Amazon SQS キューからデータを受け取るアプリケーションの相対パスを指定します。このデータは HTTP POST メッセージのメッセージ本文に挿入されます。デフォルト値は `/` です。
- [MIME タイプ] – HTTP POST メッセージで使用される MIME タイプを示します。デフォルト値は `application/json` です。ただし、独自の MIME タイプを作成し、指定できるため、どのような値でも有効になります。
- [HTTP 接続] – Amazon EC2 インスタンス内でデーモンが任意のアプリケーションに同時接続できる最大数を指定します。デフォルトは **50** です。**1** ~ **100** を指定できます。
- [可視性タイムアウト] – Amazon SQS キューからの着信メッセージが処理のためにロックされる時間を秒数で示します。設定した時間が経過すると、メッセージが再びキューに表示され、他のデーモンが読み込めるようになります。アプリケーションがメッセージ処理に必要とする予測秒数より長い値 (最大 **43200** 秒) を選択します。
- [エラー可視性タイムアウト] – 明示的なエラーで処理が失敗した後、Elastic Beanstalk が Amazon SQS キューにメッセージを返すまでの経過時間を秒数で示します。**0** ~ **43200** 秒を指定できます。

[Advanced options (アドバンスドオプション)] セクションで、以下の操作を行います。

- [最大再試行回数] - メッセージがデッドレターキューに移動されるまでに Elastic Beanstalk が Amazon SQS キューにメッセージを送信する試行回数の最大数を指定します。デフォルト値は **10** です。**1** ~ **100** を指定できます。

> **Note**
>
> [Max retries] (最大再試行回数) のオプションは、デッドレターキューで設定された Amazon SQS キューにのみ適用されます。デッドレターキューで設定されていない Amazon SQS キューの場合、Amazon SQS はメッセージをキューに保持し、[Retention period] (保持期間) オプションで指定された期間が終了するまでこれを処理します。

- [接続タイムアウト] – アプリケーションに正常に接続するまでの待機時間を秒数で示します。デフォルト値は **5** です。**1** ~ **60** 秒を指定できます。

- [アイドル状態のタイムアウト] – アプリケーションへの既存の接続が応答するまでの待機時間を秒数で示します。デフォルト値は **180** です。**1** ~ **36000** 秒を指定できます。
- [保持期間] – メッセージが有効であり、アクティブに処理される時間を秒数で示します。デフォルト値は **345600** です。**60** ~ **1209600** 秒を指定できます。

既存の Amazon SQS キューを使用する場合、ワーカー環境の作成時に行った設定が、Amazon SQS で直接行った設定と競合することがあります。たとえば、ワーカー環境の `RetentionPeriod` の値を、Amazon SQS で設定した `MessageRetentionPeriod` の値より高く設定した場合、`MessageRetentionPeriod` が経過すると、メッセージは Amazon SQS によって削除されます。

また、ワーカー環境で設定した `RetentionPeriod` の値が Amazon SQS で設定した `MessageRetentionPeriod` の値より低い場合、Amazon SQS がメッセージを削除する前にデーモンがメッセージを削除します。`VisibilityTimeout` については、ワーカー環境で設定したデーモンの値が Amazon SQS の `VisibilityTimeout` の設定をオーバーライドします。Elastic Beanstalk の設定と Amazon SQS の設定を比較して、メッセージが適切に削除されたことを確認してください。

Elastic Beanstalk 環境間のリンクの作成

アプリケーションが大きくなり、複雑化するにつれて、異なる開発性と運用のライフサイクルのコンポーネント別に分割することが必要になる場合があります。明確に定義されたインターフェイスを介して相互に対話する小規模なサービスを実行することで、各チームは独立して作業を進めることができ、デプロイのリスクを軽減できます。 AWS Elastic Beanstalk では、相互に依存するコンポーネント間で情報を共有するために環境をリンクできます。

> **Note**
>
> Elastic Beanstalk は現在、Multicontainer Docker を除くすべてプラットフォームの環境リンクをサポートしています。

環境リンクを使用すると、アプリケーションのコンポーネント環境間の接続を、名前を付けたリファレンスとして指定できます。リンクを定義する環境を作成すると、リンクと同じ名前の環境変数が Elastic Beanstalk によって設定されます。この変数の値は、ウェブサーバーあるいはワーカー環境といったその他のコンポーネントに接続するための使用できるエンドポイントです。

たとえば、アプリケーションが、フロントエンドによって収集された電子メールアドレスにウェルカムメールを送付するためにその電子メールアドレスとワーカーを収集するフロントエンドである場合、お手持ちのフロントエンドでワーカーへのリンクを作成すると、フロントエンドが自動的にワーカーのエンドポイント（キュー URL）を検出するようにできます。

アプリケーションソースのルートの `env.yaml` 名の YAML 形式のファイルで、ほかの環境へのリンクを環境マニフェストで定義します。次のマニフェストはワーカーという環境へのリンクを定義します。

~/workspace/my-app/frontend/env.yaml

```
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentLinks:
  "WORKERQUEUE": "worker"
```

上記の環境マニフェストを含むアプリケーションバージョンで環境を作成する場合、Elastic Beanstalk は同じアプリケーションに属する `worker` という環境を検索します。その環境が存在する場合、Elastic Beanstalk は `WORKERQUEUE` という環境プロパティを作成します。`WORKERQUEUE` の値は、Amazon SQS キューの URL です。フロントエンドアプリケーションは環境変数と同様にこのプロパティを読み取ることができます。詳細については、「マニフェスト環境 (`env.yaml`)」を参照してください。

環境リンクを使用するには、環境マニフェストをアプリケーションソースに追加し、EB CLI AWS CLI または SDK を使用してアップロードします。 AWS CLI または SDK を使用する場合は、 を呼び出すときに `process`フラグを設定します`CreateApplicationVersion`。

```
$ aws elasticbeanstalk create-application-version --process --application-name
 my-app --version-label frontend-v1 --source-bundle S3Bucket="amzn-s3-demo-
bucket",S3Key="front-v1.zip"
```

このオプションを使用すると、アプリケーションバージョンを作成する際にソースバンドルの環境マニフェストと設定ファイルを確認するように Elastic Beanstalk に指定できます。プロジェクトディレクトリに環境マニフェストがある場合には、EB CLI はこのフラグを自動的に設定します。

いずれかのクライアントを使用して環境を通常通りに作成します。環境を終了する必要があるときは、リンクがある環境を最初に終了します。環境が別の環境からリンクされている場合、Elastic Beanstalk はリンクされた環境の終了を防ぎます。この保護を無効にするには、`ForceTerminate` フラグを使用します。このパラメータは、 AWS CLI で `--force-terminate` として使用できます。

```
$ aws elasticbeanstalk terminate-environment --force-terminate --environment-name
 worker
```

Elastic Beanstalk 環境を無効な状態から復旧する

このトピックでは、無効な状態の Elastic Beanstalk 環境のトラブルシューティング方法を説明する背景情報とリソースをいくつか紹介します。

エラーへの対応

無効な状態の環境に対する標準オペレーションは正常に完了しません。失敗したオペレーションは、次のテキストを含むエラーを返します。

```
The stack stack_id associated with environment environment-ID is in stack-status state.
```

このエラーをトラブルシューティングして解決するには、ナレッジセンターの記事「Elastic Beanstalk 環境が無効な状態であるのはなぜですか？」を参照してください。

> **Note**
>
> 2024 年 12 月 16 日より前は、失敗したオペレーションによって次のエラーが返されました。この場合、修正アクションの完了後に環境ステータスをリセットするには、 AWS サポートに連絡する必要がありました。 `Environment is in an invalid state for this operation. Must be ready.`
> 現在も、参照されているナレッジセンターの記事の指示に従ってスタックの問題を解決する必要があります。ただし、修正アクションが正常に完了すると、Elastic Beanstalk は環境のステータスを無効から使用可能に自動的に更新し、さらに遅延することなく環境で標準オペレーションを再開できます。

エラーが発生する理由

Elastic Beanstalk にアプリケーションをデプロイすると、サービスは基盤となる AWS CloudFormation スタックを作成します。Elastic Beanstalk は、 AWS CloudFormation サービスを呼び出して環境内のリソースを起動し、設定の変更を伝達します。

Elastic Beanstalk が必要なリソースにアクセスせずに環境でオペレーションを実行すると、環境の基盤となる CloudFormation スタックが失敗状態になる可能性があります。その他の問題もこの状態

につながる可能性がありますが、アクセス許可の問題が主な原因です。スタックの失敗状態の結果として、 AWS CloudFormation は Elastic Beanstalk オペレーションリクエストがスタックの更新をさらに実行することをブロックし、UpdateEnvironment や RetrieveEnvironmentInfo などの Elastic Beanstalk オペレーションが失敗します。

この時点で、CloudFormation スタックを修正するには、まず根本的な問題の根本原因を修正する必要があります。次に、Elastic Beanstalk サービスは CloudFormation スタックのステータス変更を検出し、その後に続いて環境を利用可能なステータスにリセットします。この時点で、さらに多くのオペレーションが正常に完了できます。

アクセス許可の問題は、通常 CloudFormation スタックと Elastic Beanstalk 環境にこの影響を引き起こしますが、out-of-bandの変更も問題を引き起こす可能性があります。

> **Important**
>
> 環境の中断を避けるため、Elastic Beanstalk サービスから環境を管理および設定するオペレーションのみを開始することを強くお勧めします。Elastic Beanstalk 以外のサービスのコンソール、CLI コマンド、または SDK を使用したリソースの変更はout-of-bandの変更であり、リソースドリフトが発生します。リソースドリフトは CloudFormation スタックのステータスに影響します。これにより、Elastic Beanstalk 環境が無効な状態になります。リソースドリフトの詳細については、「 AWS CloudFormation ユーザーガイド」の「ドリフトとは」を参照してください。

Elastic Beanstalk 環境の設定

このトピックでは、Elastic Beanstalk コンソールで使用できる設定オプションに焦点を当てます。 AWS Elastic Beanstalk には、環境内のリソースをカスタマイズするための幅広いオプションと、Elastic Beanstalk の動作とプラットフォーム設定が用意されています。

次のトピックでは、コンソールで環境を設定する方法について説明します。また、設定ファイルや API 設定オプションで使用するコンソールオプションに対応する基本的な名前空間についても説明します。高度な設定方法の詳細については、「環境の設定 (アドバンスト)」を参照してください 。

トピック

- プロビジョニングされたリソース
- Elastic Beanstalk コンソールを使用した環境設定
- お客様のElastic Beanstalk 環境に対する Amazon EC2 インスタンス
- Elastic Beanstalk 環境インスタンスの Auto Scaling
- Elastic Beanstalk 環境のロードバランサー
- Elastic Beanstalk 環境にデータベースを追加する
- AWS Elastic Beanstalk 環境セキュリティ
- Elastic Beanstalk 環境でのリソースのタグ付け
- 環境変数およびその他のソフトウェア設定
- Amazon SNS を使用した Elastic Beanstalk 環境の通知
- Elastic Beanstalk を使用した Amazon Virtual Private Cloud (Amazon VPC) の設定
- Elastic Beanstalk 環境のドメイン名

プロビジョニングされたリソース

ウェブサーバー環境を作成すると、Elastic Beanstalk は、アプリケーションのオペレーションをサポートする複数のリソースを作成します。この章では、Elastic Beanstalk 環境でこれらのリソースをカスタマイズする方法について説明します。

- EC2 インスタンス – 選択したプラットフォームでウェブ・アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを

実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。

- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。
> Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

Elastic Beanstalk コンソールを使用した環境設定

このトピックでは、Elastic Beanstalk コンソールで使用可能な設定オプションの概要と、設定ページの操作方法について説明します。

環境の設定の要約を表示するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。

設定ページ

[Configuration overview (設定の概要)] ページに、一連の設定カテゴリが表示されます。各設定カテゴリは、関連する一連のオプションをグループ化したものです。

サービスアクセス

このカテゴリのオプションでは、Elastic Beanstalk が環境の管理に使用するサービスロールと EC2 インスタンスプロファイルを選択します。必要に応じて、EC2 キーペアを選択して EC2 インスタンスに安全にログインします。

ネットワークとデータベース

このカテゴリのオプションは、VPC 設定を構成し、環境の EC2 インスタンスとロードバランサーのサブネットを設定します。また、環境と統合された Amazon RDS データベースを設定するオプションも提供します。

インスタンストラフィックとスケーリング

これらのオプションは、環境の EC2 インスタンスの容量、スケーリング、ロードバランシングをカスタマイズします。ロードバランサーに送信されたリクエストの詳細情報のログを取得するように Elastic Load Balancing を設定することもできます。

次のオプションも EC2 インスタンスの設定に使用できます。

- ルートボリュームタイプ、サイズ、入出力オペレーションレート (IOPS)、スループット。
- インスタンスメタデータサービス (IMDS) の有効化。
- インスタンスのトラフィックを制御する EC2 セキュリティグループの選択。
- CloudWatch メトリクスのモニタリング間隔。
- メトリクスログ記録の時間間隔。

更新、モニタリング、ログ記録

このカテゴリは、次のオプションを設定します。

- 環境ヘルスレポート (拡張ヘルスレポートを選択するオプションを含む)。
- Elastic Beanstalk が環境に変更をデプロイするタイミングと方法を定義するマネージドプラットフォーム更新。
- アプリケーションの動作に関するデータを収集して問題や最適化の機会を特定するための X-Ray サービスの有効化。
- プロキシサーバーや OS 環境のプロパティなど、プラットフォーム固有のオプション。

設定ページのナビゲーション

設定カテゴリの [編集] を選択して関連する設定ページを表示すると、オプションの値をすべて確認し、変更することができます。

設定カテゴリのナビゲーション

次のいずれかのアクションを使用して、設定カテゴリページに移動します。

- キャンセル – 設定の変更を適用せずに、[設定] 概要ページに戻ります。[キャンセル] を選択すると、設定カテゴリで行った保留中の変更がコンソールから失われます。

  [イベント] や [ログ] など、左側のナビベーションページで別の項目を選択することによって、設定変更をキャンセルすることもできます。
- [続行] – [設定の概要] ページに戻ります。変更を続けるか、保留中の変更を適用できます。
- [適用] - 設定カテゴリで行った変更を環境に適用します。場合によっては、設定の決定の 1 つの結果を確認するように求められることがあります。

[設定] 概要ページのナビゲーション

設定カテゴリの [編集] を選択して関連する設定ページを表示すると、オプションの値をすべて確認し、変更することができます。オプションを確認し、変更したら、[設定] 概要ページから次のいずれかのアクションを選択できます。

- [キャンセル] – 設定変更を適用せずに、環境のダッシュボードに戻ります。[キャンセル] を選択すると、設定カテゴリで行った保留中の変更がコンソールから失われます。

  [イベント] や [ログ] など、左側のナビベーションページで別の項目を選択することによって、設定変更をキャンセルすることもできます。
- [変更の確認] – 設定カテゴリで行った保留中のすべての変更の概要が表示されます。詳細については、「[変更の確認] ページ」を参照してください。
- [Apply changes] (変更の適用) - 設定カテゴリで行った変更を環境に適用します。場合によっては、設定の決定の 1 つの結果を確認するように求められることがあります。

[変更の確認] ページ

[変更の確認] ページには、設定カテゴリで行い、環境にまだ適用されていない保留中のオプション変更のすべてを示すテーブルが表示されます。

テーブルでは、各オプションが [Namespace] (名前空間) と [Option] (オプション) の組み合わせとして一覧表示され、これらによって Elastic Beanstalk はオプションを識別します。詳細については、「設定オプション」を参照してください。

変更を確認したら、次のいずれかのアクションを選択できます。

- [続行] – [設定の概要] ページに戻ります。変更を続けるか、保留中の変更を適用できます。
- [Apply changes] (変更の適用) - 設定カテゴリで行った変更を環境に適用します。場合によっては、設定の決定の 1 つの結果を確認するように求められることがあります。

お客様のElastic Beanstalk 環境に対する Amazon EC2 インスタンス

このトピックでは、Elastic Beanstalk 環境に影響する Amazon EC2 インスタンスと設定オプションについて説明します。

ウェブサーバー環境を作成すると、 はインスタンスと呼ばれる 1 つ以上の Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン AWS Elastic Beanstalk を作成します。

環境内のインスタンスは、選択したプラットフォームでウェブアプリケーションを実行するように設定されます。環境の作成中、または既に実行を開始してから、環境のインスタンスのさまざまなプロパティと動作を変更できます。これらの変更は、環境にデプロイするソースコードを変更することでも、行うことができます。詳細については、 the section called “設定オプション” を参照してください。

> **Note**
>
> 環境内の Auto Scaling グループは、アプリケーションを実行する Amazon EC2 インスタンスを管理します。このトピックで説明されている設定を変更すると、起動設定も変更されます。起動設定は、Amazon EC2 起動テンプレートまたは、Auto Scaling グループ起動設定のリソースです。この変更にはすべてのインスタンスの置換が必要です。また、ローリング更新またはイミュータブルな更新のいずれかが設定されている場合、そのいずれかをトリガーします。

EC2 インスタンス購入オプション

Elastic Beanstalk は、いくつかの Amazon EC2 インスタンス購入オプションをサポートしています。

- オンデマンド – オンデマンドインスタンスは、従量制料金のリソースであり、使用時に長期契約は必要ありません。
- リザーブド – リザーブドインスタンスは、お客様の環境にマッチするオンデマンドインスタンスに自動的に適用される事前購入割引課金です。
- スポット – スポットインスタンスは、未使用の Amazon EC2 インスタンスで、オンデマンドより低価格で利用できます。環境内のスポットインスタンスの割り当てを有効にして設定できます。詳細については、「Elastic Beanstalk 環境インスタンスの Auto Scaling 」を参照してください。

トピック

- Amazon EC2 インスタンスタイプ
- Elastic Beanstalk コンソールを使用した Amazon EC2 インスタンスの設定
- EC2 セキュリティグループの管理
- を使用した Amazon EC2 セキュリティグループとインスタンスタイプの設定 AWS CLI

- 名前空間オプションを使用した Amazon EC2 インスタンスの設定
- Elastic Beanstalk 環境のインスタンスでの IMDS の設定

Amazon EC2 インスタンスタイプ

このトピックでは、インスタンスタイプという用語について説明します。新しい環境を作成すると、Elastic Beanstalk は Amazon EC2 に基づく Amazon EC2 インスタンスをプロビジョニングします。選択するのは、インスタンスタイプです。選択したインスタンスタイプによって、インスタンスを実行するホストハードウェアが決まります。EC2 インスタンスタイプは、それぞれのプロセッサアーキテクチャに基づいて分類できます。Elastic Beanstalk は、次のプロセッサアーキテクチャに基づくインスタンスタイプをサポートしています。 AWS Graviton 64 ビット Arm アーキテクチャ (arm64)、64 ビットアーキテクチャ (x86)、32 ビットアーキテクチャ (i386)。Elastic Beanstalk は新しい環境を作成するときにデフォルトで x86 プロセッサアーキテクチャを選択します。

> **Note**
>
> i386 32 ビットアーキテクチャは、大半の Elastic Beanstalk プラットフォームで support されなくなりました。代わりに x86 または arm64 アーキテクチャタイプを選択することをお勧めします。Elastic Beanstalk設定オプションの i386 プロセッサインスタンスタイプの場合`aws:ec2:instances`名前空間。

特定の Elastic Beanstalk 環境の設定に含まれるインスタンスタイプはすべて、同じタイプのプロセッサアーキテクチャを持つ必要があります。x86 アーキテクチャに基づく t2.ミディアム インスタンスタイプが既にある既存の環境に、新しいインスタンスタイプを追加するとします。t2.small など、同じアーキテクチャの別のインスタンスタイプのみを追加できます。既存のインスタンスタイプを別のアーキテクチャのインスタンスタイプに置き換えることもできます。ただし、コマンド内のすべてのインスタンスタイプが同じタイプのアーキテクチャに基づいていることを確認してください。

Amazon EC2 がインスタンスタイプを導入すると、Elastic Beanstalk は互換性のあるインスタンスタイプの support を、いつも追加します。利用可能なインスタンスタイプについては、「Amazon EC2 ユーザーガイド」の「インスタンスタイプ」を参照してください。

> **Note**
>
> Elastic Beanstalk では、Graviton がサポートされているすべてのリージョンで、最新のすべての Amazon Linux 2 プラットフォームで AWS Graviton がサポートされるようになりまし

た。arm64 ベースのインスタンスタイプを使用した Elastic Beanstalk 環境の作成の詳細については、を参照してください。Elastic Beanstalk コンソールを使用した Amazon EC2 インスタンスの設定。
arm64 アーキテクチャで Amazon EC2 インスタンスを実行する新しい環境を作成し、デプロイオプションElastic Beanstalk
Graviton arm64 ベースのプロセッサの詳細については、以下の AWS リソースを参照してください。

- メリット — AWS Graviton プロセッサ
- 開始方法と言語固有の考慮事項などのその他のトピック — Graviton GitHub AWS の開始方法の記事 GitHub

Elastic Beanstalk コンソールを使用した Amazon EC2 インスタンスの設定

Elastic Beanstalk コンソールで、Elastic Beanstalk 環境の Amazon EC2 インスタンスを設定または変更できます。

> **Note**
> Elastic Beanstalk コンソールには、既存の環境のプロセッサアーキテクチャを変更するオプションがありませんが、 AWS CLIを使用して変更できます。コマンドの例については、「を使用した Amazon EC2 セキュリティグループとインスタンスタイプの設定 AWS CLI」を参照してください。

環境を構築する間にElastic Beanstalk コンソールで Amazon EC2 インスタンスを設定するには

1. Elastic Beanstalk 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 [Environments (環境)] を選択します。
3. [新しい環境の作成] を選択して、環境の作成を開始します。
4. ウィザードのメインページで、[環境の作成] を選択する前に、[さらにオプションを設定] を選択します。
5. [インスタンス] 設定カテゴリで、[編集] を選択します。このカテゴリの設定を変更し、[Apply (適用)] を選択します。設定の説明については、このページのセクション the section called “インスタンスカテゴリの設定” を参照してください。

6. [キャパシティー] 設定カテゴリで、[編集] を選択します。このカテゴリの設定を変更し、[Continue (続行)] を選択します。設定の説明については、このページのセクション the section called “キャパシティーカテゴリの設定” を参照してください。

> **プロセッサアーキテクチャの選択**
>
> [] まで下にスクロールします。プロセッサをクリックして、EC2 インスタンスのプロセッサアーキテクチャを選択します。コンソールには、で以前に選択したプラットフォームでサポートされているプロセッサアーキテクチャが一覧表示されます。環境を作成するパネル。
> 必要なプロセッサアーキテクチャが表示されない場合は、設定のカテゴリリストに戻り、それをサポートするプラットフォームを選択します。からの容量の変更パネル、選択しますキャンセル。次に、プラットフォームのバージョンを変更するをクリックして、新しいプラットフォーム設定を選択します。次に、容量設定のカテゴリを選択します。編集プロセッサー・ アーキテクチャーの選択をもう一度見てください。

7. [保存] を選択し、環境に必要なその他の任意の設定変更を行います。
8. [環境を作成] を選択します。

Elastic Beanstalk コンソールで実行環境の Amazon EC2 インスタンス を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [インスタンス] 設定カテゴリで、[編集] を選択します。このカテゴリの設定を変更し、[Apply (適用)] を選択します。設定の説明については、このページのセクション the section called “インスタンスカテゴリの設定” を参照してください。
5. [キャパシティー] 設定カテゴリで、[編集] を選択します。このカテゴリの設定を変更し、[Continue (続行)] を選択します。設定の説明については、このページのセクション the section called “キャパシティーカテゴリの設定” を参照してください。

インスタンスカテゴリの設定

Amazon EC2 インスタンスに関連する次の設定は、[Instances (インスタンス)] 設定カテゴリで使用できます。

オプション

- 間隔のモニタリング
- ルートボリューム (起動デバイス)
- インスタンスメタデータサービス
- EC2 セキュリティグループ

間隔のモニタリング

デフォルトでは、環境内のインスタンスは、ベーシックヘルスメトリクスを 5 分間隔で Amazon CloudWatch に公開します (追加料金は発生しません)。

より詳細なレポートを実行するには、[Monitoring interval] (モニタリング間隔) を [1 minute] (1 分) に設定し、環境内のリソースがベーシックヘルスのメトリクスを CloudWatch に公開する頻度を高めます。この場合、CloudWatch の利用料金は、1 分間隔のメトリクスに適用されます。詳細については、「Amazon CloudWatch」を参照してください。

ルートボリューム (起動デバイス)

環境内の各インスタンスは、ルートボリュームを使用して設定されます。ルートボリュームは、オペレーティングシステムやライブラリ、スクリプト、アプリケーションのソースコードを保存するためにインスタンスにアタッチされる Amazon EBS ブロックデバイスです。デフォルトでは、すべてのプラットフォームでストレージ用の汎用 SSD ブロックデバイスが使用されます。

[ルートボリュームタイプ] を変更して、マグネティックストレージやプロビジョンド IOPS SSD ボリュームタイプを使用すれば、必要に応じてボリュームサイズを増大することが可能です。プロビジョンド IOPS ボリュームでは、プロビジョニングする IOPS の数を選択する必要があります。スループットは gp3 SSD ボリュームタイプにのみ適用されます。プロビジョンに必要なスループットを入力できます。1 秒あたり 125 ~ 1000 メビバイト (MiB/s) の範囲に及びます。必要なパフォーマンスと価格の要件を満たすボリュームタイプを選択します。

> **Important**
>
> `RootVolumeType` オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してください Elastic Beanstalk 環境を起動テンプレートに移行する 。

詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EBS ボリュームの種類」、および Amazon EBS 製品の詳細に関するページを参照してください。

インスタンスメタデータサービス

インスタンスメタデータサービス (IMDS) は、インスタンス上のコードによって、インスタンスメタデータに安全にアクセスするために使用されるインスタンス上のコンポーネントです。コードは、次の二つのうち一つのメソッドを使って、実行中のインスタンスからインスタンスのメタデータにアクセスできます。二つのメソッドは、インスタンスのメタデータサービスバージョン1(IMDSv1)か、インスタンスのメタデータサービスバージョン2(IMDSv2)です。IMDSv2 の方が安全性に優れています。IMDSv1 を無効にして、IMDSv2 を適用します。詳細については、「the section called “IMDS”」を参照してください。

> **Note**
>
> この設定ページの IMDS セクションは、IMDSv2 をサポートするプラットフォームバージョンに対してのみ表示されます。

EC2 セキュリティグループ

インスタンスにアタッチされているセキュリティグループは、インスタンスに到達および終了できるトラフィックを決定します。

Elastic Beanstalk が作成するデフォルトの EC2 セキュリティグループでは、HTTP (80) および SSH(22) の標準ポートで、インターネットまたはロードバランサーからのすべての着信トラフィックが許可されます。EC2 インスタンスのファイアウォールルールを指定するために、独自のカスタムセキュリティグループを定義することもできます。セキュリティグループは、他のポートまたは

他のソースからのトラフィックを許可できます。例えば、制限された IP アドレス範囲から ポート 22 に着信したトラフィックを許可する SSH アクセス用セキュリティグループを作成できます。または、セキュリティを強化するために、自分だけがアクセスできる踏み台ホストからのトラフィックを許可するものを作成できます。

aws:autoscaling:launchconfiguration 名前空間の `DisableDefaultEC2SecurityGroup`オプションを に設定することで、デフォルトの EC2 セキュリティグループから環境をオプトアウトするように選択できます`true`。このオプションは コンソールでは使用できませんが、 で設定できます AWS CLI。詳細については、「EC2 セキュリティグループの管理」を参照してください。

セキュリティグループの詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 セキュリティグループ」を参照してください。

> **Note**
>
> 環境 A のインスタンスと環境 B のインスタンス間のトラフィックを許可するには、Elastic Beanstalk が環境 B にアタッチしたセキュリティグループにルールを追加します。そうすると、Elastic Beanstalk が環境 A にアタッチしたセキュリティグループを指定できます。これにより、環境 A のインスタンスとの間で着信または送信されるトラフィックが許可されます。ただし、これを行うことで 2 つのセキュリティグループ間に依存性が生じます。後で環境 A を終了しようとすると、Elastic Beanstalk は環境 B のセキュリティグループが依存しているため、環境のセキュリティグループを削除できなくなります。
> したがって、最初に別のセキュリティグループを作成することをお勧めします。次に、環境 A にアタッチして、環境 B のセキュリティグループのルールでその環境を指定します。

キャパシティーカテゴリの設定

Amazon EC2 インスタンスに関連する次の設定は、[Capacity (キャパシティー)] 設定カテゴリで使用できます。

オプション

- インスタンスのタイプ
- AMI ID

インスタンスのタイプ

Instance type (インスタンスタイプ) の設定によって、アプリケーションを実行する時に起動する Amazon EC2 インスタンスのタイプが決まります。この設定ページには、インスタンスタイプ。1 つ以上のインスタンスタイプを選択できます。Elastic Beanstalk コンソールには、環境用に設定されたプロセッサアーキテクチャに基づくインスタンスタイプのみが表示されます。したがって、同じプロセッサアーキテクチャのインスタンスタイプのみを追加できます。

> **Note**
>
> Elastic Beanstalk コンソールには、既存の環境のプロセッサアーキテクチャを変更するオプションがありませんが、 AWS CLIを使用して変更できます。コマンドの例については、「を使用した Amazon EC2 セキュリティグループとインスタンスタイプの設定 AWS CLI」を参照してください。

インスタンスは、負荷がかかった状態でもアプリケーションを実行できる性能があるものを選択します。ただし、性能が高すぎて、ほとんどの時間アイドル状態になるようなインスタンスは避けてください。開発目的のため、t2 インスタンスファミリーは、短期間のバーストが可能な中程度の強度を備えています。大規模なアプリケーションの可用性を高くするには、インスタンスプールを使用して、インスタンスの 1 つが停止しても容量が大きな影響を受けないようにします。まず、平常時に中程度の負荷下で 5 つのインスタンスを実行できるインスタンスタイプから開始します。いずれかのインスタンスが失敗した場合は、残りのインスタンスがその他のトラフィックを吸収します。また、ピーク時間帯のトラフィック上昇に対応するため、環境を拡張する時間としてキャパシティーバッファが設けられています。

Amazon EC2 インスタンスファミリーとタイプの詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスタイプ」を参照してください。要件を満たすインスタンスタイプとそのサポートされているリージョンを確認するには、「Amazon EC2 ユーザーガイド」の「利用可能なインスタンスタイプ」を参照してください。

AMI ID

Amazon マシンイメージ (AMI) は、Amazon Linux または Windows Server のマシンイメージで、環境で Amazon EC2 インスタンスを起動する目的で Elastic Beanstalk が使用します。Elastic Beanstalk は、アプリケーションの実行に必要なツールとリソースを含むマシンイメージを提供します。

Elastic Beanstalk は、選択したリージョン、プラットフォームのバージョンおよび、プロセッサ・アーキテクチャに基づいた環境のデフォルト AMI を選択します。カスタム AMI を作成した場合は、デフォルトの AMI ID とそれを置き換えます。

EC2 セキュリティグループの管理

Elastic Beanstalk が環境を作成すると、その環境で起動される EC2 インスタンスにデフォルトのセキュリティグループが割り当てられます。インスタンスにアタッチされているセキュリティグループは、インスタンスに到達および終了できるトラフィックを決定します。

Elastic Beanstalk が作成するデフォルトの EC2 セキュリティグループでは、HTTP (80) および SSH(22) の標準ポートで、インターネットまたはロードバランサーからのすべての受信トラフィックが許可されます。EC2 インスタンスのファイアウォールルールを指定するために、独自のカスタムセキュリティグループを定義することもできます。セキュリティグループは、他のポートまたは他のソースからのトラフィックを許可できます。例えば、制限された IP アドレス範囲から ポート 22 に着信したトラフィックを許可する SSH アクセス用セキュリティグループを作成できます。または、セキュリティを強化するために、自分だけがアクセスできる踏み台ホストからのトラフィックを許可するものを作成できます。

aws:autoscaling:launchconfiguration 名前空間の `DisableDefaultEC2SecurityGroup`オプションを に設定することで、デフォルトの EC2 セキュリティグループから環境をオプトアウトするように選択できます`true`。AWS CLI または 設定ファイルを使用して、このオプションを環境に適用し、EC2 インスタンスにカスタムセキュリティグループをアタッチします。

マルチインスタンス環境での EC2 セキュリティグループの管理

マルチインスタンス環境でカスタム EC2 セキュリティグループを作成する場合は、ロードバランサーと受信トラフィックルールがインスタンスを安全かつアクセス可能にする方法も考慮する必要があります。

複数の EC2 インスタンスを持つ環境へのインバウンドトラフィックはロードバランサーによって管理され、すべての EC2 インスタンス間で受信トラフィックをルーティングします。Elastic Beanstalk がデフォルトの EC2 セキュリティグループを作成すると、ロードバランサーからの受信トラフィックを許可するインバウンドルールも定義されます。セキュリティグループにこのインバウンドルールがないと、受信トラフィックはインスタンスに入ることができません。この条件は、基本的に外部リクエストからインスタンスをブロックします。

ロードバランシングされた環境のデフォルトの EC2 セキュリティグループを無効にすると、Elastic Beanstalk はいくつかの設定ルールを検証します。設定が検証チェックを満たさない場合、必要な設定を指定するように指示するメッセージが表示されます。検証チェックは次のとおりです。

- アプリケーションロードバランサーかクラシックロードバランサーかに応じてaws:elb:loadbalancer、 aws:elbv2:loadbalancerまたは `SecurityGroups`のオプションを使用して、ロードバランサーに少なくとも 1 つのセキュリティグループを割り当てる必要があります。AWS CLI 例については、「」を参照してください AWS CLIを使用した設定。
- EC2 インスタンスがロードバランサーからトラフィックを受信できるようにするインバウンドトラフィックルールが存在している必要があります。EC2 セキュリティグループとロードバランサーセキュリティグループの両方が、これらのインバウンドルールを参照する必要があります。詳細については、次の「トラフィックのインバウンドルール」セクションを参照してください。

トラフィックのインバウンドルール

マルチインスタンス環境の EC2 セキュリティグループ (複数可) には、ロードバランサーセキュリティグループを参照するインバウンドルールを含める必要があります。これは、任意のタイプのロードバランサー、専用または共有の環境、およびカスタムまたはデフォルトのロードバランサーセキュリティグループを持つ環境に適用されます。

EC2 コンソールで、環境コンポーネントにアタッチされているすべてのセキュリティグループを表示できます。次の図は、環境の作成オペレーション中に Elastic Beanstalk がデフォルトで作成するセキュリティグループの EC2 コンソールリストを示しています。

セキュリティグループ画面には、環境と関連するセキュリティグループが表示されます。GettingStarted-env と GettingStarted3-env はどちらも、専用のロードバランサーを備えたマルチインスタンス環境です。これらの各環境には、EC2 インスタンス用とロードバランサー用の EC2 つのセキュリティグループが一覧表示されています。Elastic Beanstalk は、環境の作成時にこれらのセキュリティグループを作成します。GettingStarted5-env にはロードバランサーセキュリティグループがありません。これは、EC2 インスタンスが 1 つしかないため、ロードバランサーがないためです。

インバウンドルール画面は、GettingStarted3-env のインスタンスの EC2 セキュリティグループにドリルダウンします。この例では、EC2 セキュリティグループのインバウンドルールを定義します。インバウンドルールのソース列には、前のイメージにリストされているロードバランサーセキュリティグループのセキュリティグループ ID がリストされていることに注意してください。このルールにより、GettingStarted3-env の EC2 インスタンスは、ポート 80 でその特定のロードバランサーからインバウンドトラフィックを受信できます。

詳細については、Amazon EC2 ユーザーガイド」の「インスタンスのセキュリティグループを変更する」およびElastic Load Balancing ルール」を参照してください。

を使用した Amazon EC2 セキュリティグループとインスタンスタイプの設定 AWS CLI

AWS コマンドラインインターフェイス (AWS CLI) を使用して、Elastic Beanstalk 環境で Amazon EC2 インスタンスを設定できます。

を使用した EC2 セキュリティグループの設定 AWS CLI

このトピックでは、単一インスタンス環境とロードバランシング (マルチインスタンス) 環境の両方の異なる EC2 セキュリティグループ設定の例を示します。これらの例のオプションの詳細については、「aws:autoscaling:launchconfiguration」を参照してください。

> **メモ**
>
> 環境の作成オペレーションは、デフォルトで EC2 セキュリティグループを提供します。また、デフォルトでアプリケーションロードバランサーを使用して環境を作成します。
> 更新環境オペレーションは、ブール値オプション を使用して、環境のデフォルトの EC2 セキュリティグループを無効または有効にするために使用できますDisableDefaultEC2SecurityGroup。例 5 は、以前に環境を変更したことがある場合は、環境をデフォルトのセキュリティ設定に戻す方法を示しています。

次の例は、デフォルトの EC2 セキュリティグループをオプトアウトし、代わりにカスタムセキュリティグループを提供する create-environment コマンドを示しています。DisableDefaultEC2SecurityGroup オプションは に設定されているためtrue、Elastic Beanstalk が通常 EC2 インスタンスに関連付けるデフォルトの EC2 セキュリティグループは作成されません。したがって、 SecurityGroupsオプションを他のセキュリティグループに提供する必要があります。

aws:elasticbeanstalk:environment EnvironmentType オプションは に設定されていることに注意してくださいSingleInstance。がLoadBalancedデフォルトの であるため、単一のインスタンス環境を作成するには、このオプションを指定する必要がありますEnvironmentType。この環境にはロードバランサーが含まれていないため、ロードバランサーのセキュリティグループを指定する必要はありません。

Example 1 — カスタム EC2 セキュリティグループを持つ新しい単一インスタンス環境 (名前空間オプションをインライン)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 applrunning Node.js 22" \
--option-settings \
Namespace=aws:elasticbeanstalk:environment,OptionName=EnvironmentType,Value=SingleInstance
 \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-
elasticbeanstalk-ec2-role \
Namespace=aws:autoscaling:launchconfiguration,OptionName=DisableDefaultEC2SecurityGroup,Value=t
 \
Namespace=aws:autoscaling:launchconfiguration,OptionName=SecurityGroups,Value=sg-
abcdef01, sg-abcdef02 \
Namespace=aws:autoscaling:launchconfiguration,OptionName=EC2KeyName,Value=my-keypair
```

別の方法として、`options.json`ファイルを使用して、インラインで名前空間オプションを含めるのではなく、名前空間オプションを指定します。

Example 2 — カスタム EC2 セキュリティグループを持つ新しい単一インスタンス環境 (**`options.json`** ファイルの名前空間オプション)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 22" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  { "Namespace" : "aws:elasticbeanstalk:environment",
    "OptionName" : "EnvironmentType",
    "Value" : "SingleInstance"
  },
```

```
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "EC2KeyName",
    "Value": "my-keypair"
  }
]
```

次の の例では、負荷分散された環境を作成します。にLoadBalancerType設定されたaws:elasticbeanstalk:environment名前空間オプションを指定しますapplication。DisableDefaultEC2SecurityGroup オプションを使用してデフォルトの EC2 セキュリティグループを無効にするため、前の例のように、aws:autoscaling:launchconfigurationSecurityGroupsオプションを使用して EC2 インスタンスに独自のカスタムセキュリティグループを再度提供する必要があります。この環境にはトラフィックをルーティングするロードバランサーがあるため、ロードバランサーのセキュリティグループも提供する必要があります。

クラシックロードバランサーを使用する環境を作成するが、それ以外の場合は同じ設定にするには、aws:elasticbeanstalk:environment名前空間オプションの設定を LoadBalancerTypeに更新しますclassic。

ロードバランサータイプごとに異なる名前空間があり、セキュリティグループを指定するオプションがあります。

- Application Load Balancer – aws:elbv2:loadbalancerSecurityGroupsオプション
- Classic Load Balancer – aws:elb:loadbalancerSecurityGroupsオプション

- Network Load Balancer – Network Load Balancer にはセキュリティグループがないため、VPC 識別子を使用して EC2 セキュリティグループを設定します。詳細については、「Network Load Balancer ユーザーガイド」の「Network Load Balancer のセキュリティグループを更新する」を参照してください。

Example 3 — カスタム EC2 セキュリティグループを持つ新しいマルチインスタンス環境 (**options.json** ファイルの名前空間オプション)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 22" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace" : "aws:elasticbeanstalk:environment",
    "OptionName" : "EnvironmentType",
    "Value" : "LoadBalanced"
  },
  {
  "Namespace" : "aws:elasticbeanstalk:environment",
    "OptionName" : "LoadBalancerType",
    "Value" : "application"
  },
  {
    "Namespace" : "aws:elbv2:loadbalancer",
    "OptionName" : "SecurityGroups",
    "Value" : "sg-abcdefghikl012345"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
```

```
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "EC2KeyName",
    "Value": "my-keypair"
  }
]
```

update-environment コマンドを使用して、既存の環境のデフォルトの EC2 セキュリティグループを無効にすることができます。次のコマンド例では、デフォルトの EC2 セキュリティグループを無効にし、環境の EC2 インスタンスのカスタム EC2 セキュリティグループを割り当てます。

環境がロードバランシングされているかどうかとロードバランサーのタイプに応じて、例 4 (a)、4 (b)、または 4 (c) のサンプル`options.jason`ファイルを使用します。設定ファイル 4 (a) は、単一インスタンス環境のセキュリティグループを指定します。ロードバランサーを必要としないため、EC2 インスタンスのセキュリティグループのみを提供します。設定ファイル 4 (b) と 4 (c) は、アプリケーションロードバランサーとクラシックロードバランサーのセキュリティグループを指定します。このような場合は、ロードバランサーのセキュリティグループも指定する必要があります。

Example 4 — 既存の環境を更新してデフォルトの EC2 セキュリティグループを無効にする (**`options.json`** ファイルの名前空間オプション)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 22" \
--option-settings file://options.json
```

Example 4 (a) — 単一インスタンス環境の設定ファイル (ロードバランサーなし)

```
### example options.json ###
[
  {
```

```
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  }
]
```

アプリケーションロードバランサーを使用する環境を更新するには、`aws:elbv2:loadbalancer`名前空間を使用してロードバランサーのセキュリティグループを指定します。

Example 4 (b) — アプリケーションロードバランサーを使用する環境の設定ファイル

```
### example options.json ###
[
  {
    "Namespace" : "aws:elbv2:loadbalancer",
    "OptionName" : "SecurityGroups",
    "Value" : "sg-abcdefghikl012345"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  }
]
```

クラシックロードバランサーを使用する環境を更新するには、 `aws:elb:loadbalancer`名前空間を使用してロードバランサーのセキュリティグループを指定します。

Example 4 (c) — クラシックロードバランサーを使用する環境の設定ファイル

```
### example options.json ###
```

```
[
  {
    "Namespace" : "aws:elb:loadbalancer",
    "OptionName" : "SecurityGroups",
    "Value" : "sg-abcdefghikl012345"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",n
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  }
]
```

Elastic Beanstalk が割り当てるデフォルトのセキュリティグループを使用して環境をデフォルトの動作と設定に戻すには、update-environment コマンドを使用して `DisableDefaultEC2SecurityGroup`を に設定します`false`。マルチインスタンス環境の場合、Elastic Beanstalk は環境のロードバランサーのセキュリティグループとネットワークトラフィックルールも処理します。

次の例は、単一インスタンス環境またはマルチインスタンス (ロードバランシング) 環境の両方に適用されます。

Example 5 — デフォルトのセキュリティグループを使用して環境を に更新する (**`options.json`** ファイルの名前空間オプション)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 22" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
```

```
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "false"
  }
]
```

を使用してインスタンスタイプで EC2 を設定する AWS CLI

このトピックでは、 環境内の EC2 インスタンスのインスタンスタイプを設定するための例を示します。

最初の 2 つの例では、新しい環境を作成します。このコマンドは、arm64 プロセッサアーキテクチャに基づく Amazon EC2 インスタンスタイプ t4g.small を指定します。Elastic Beanstalk は、リージョン、プラットフォームのバージョン、インスタンスタイプに基づいて EC2 インスタンスのイメージ ID (AMI) をデフォルト設定します。インスタンスタイプは、プロセッサアーキテクチャに対応します。`-solution-stack-name`パラメータはプラットフォームバージョンに適用されます。

Example 1 — 新しい arm64 ベースの環境を作成する (名前空間オプションをインライン)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-elasticbeanstalk-ec2-role \
Namespace=aws:ec2:instances,OptionName=InstanceTypes,Value=t4g.small
```

別の方法として、`options.json`ファイルを使用して、インラインで名前空間オプションを含めるのではなく、名前空間オプションを指定します。

Example 2 — 新しい arm64 ベースの環境 (名前空間オプション) を作成します。**`options.json`**ファイル)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
```

```
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:ec2:instances",
    "OptionName": "InstanceTypes",
    "Value": "t4g.small"
  }
]
```

次の 2 つの例では、既存の環境の設定をupdate-environmentコマンド。この例では、arm64 プロセッサアーキテクチャに基づく別のインスタンスタイプを追加します。既存の環境では、追加されるすべてのインスタンスタイプが同じプロセッサアーキテクチャを持つ必要があります。既存のインスタンスタイプを別のアーキテクチャのインスタンスタイプに置き換えることもできます。ただし、コマンド内のすべてのインスタンスタイプが同じタイプのアーキテクチャであることを確認します。

Example 3 — 既存の arm64 ベースの環境を更新する (名前空間オプションをインライン)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-
elasticbeanstalk-ec2-role \
Namespace=aws:ec2:instances,OptionName=InstanceTypes,Value=t4g.small,t4g.micro
```

別の方法として、options.jsonファイルを使用して、インラインで名前空間オプションを含めるのではなく、名前空間オプションを指定します。

Example 4 — 既存の arm64 ベースの環境を更新する (名前空間オプション)**options.jsonファイル**)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:ec2:instances",
    "OptionName": "InstanceTypes",
    "Value": "t4g.small, t4g.micro"
  }
]
```

次の 2 つの例で詳しく説明します。環境を作成するコマンド。これらの例は、の値を提供しません。InstanceTypes。メトリックInstanceTypes値は指定されていません。Elastic Beanstalk はデフォルトで x86 ベースのプロセッサアーキテクチャです。環境の EC2 インスタンスのイメージ ID (AMI) は、リージョン、プラットフォームのバージョン、およびデフォルトのインスタンスタイプに従ってデフォルト設定されます。インスタンスタイプは、プロセッサアーキテクチャに対応します。

Example 5 — 新しい x86 ベースの環境を作成する (名前空間オプションをインライン)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
```

```
--option-settings \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-elasticbeanstalk-ec2-role
```

別の方法として、options.jsonファイルを使用して、インラインで名前空間オプションを含めるのではなく、名前空間オプションを指定します。

Example 6 — 新しい x86 ベースの環境 (名前空間オプション) を作成します。**options.jsonファイル**)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  }
]
```

名前空間オプションを使用した Amazon EC2 インスタンスの設定

aws:autoscaling:launchconfiguration 名前空間で、設定のオプションを使用して、コンソールで使用できない追加のオプションを含む、環境のインスタンスを設定できます。

> **Important**
>
> DisableIMDSv1、RootVolumeType、または BlockDeviceMappingsオプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可

能性があります。詳細およびその他の考慮事項については、「」を参照してくださいElastic Beanstalk 環境を起動テンプレートに移行する 。

次の設定ファイルの例では、このトピックで説明されている基本的な設定オプションを使用します。ロードバランサーのセキュリティグループを指定する必要がある場合の追加設定オプションの例については、「」を参照してください AWS CLIを使用した設定。

```
option_settings:
  aws:autoscaling:launchconfiguration:
    SecurityGroups: my-securitygroup
    MonitoringInterval: "1 minute"
    DisableIMDSv1: false
    DisableDefaultEC2SecurityGroup: true
    SecurityGroups: "sg-abcdef01, sg-abcdef02"
    EC2KeyName: my-keypair
    IamInstanceProfile: "aws-elasticbeanstalk-ec2-role"
    BlockDeviceMappings: "/dev/sdj=:100,/dev/sdh=snap-51eef269,/dev/sdb=ephemeral0"
  aws:elasticbeanstalk:environment:
    EnvironmentType: SingleInstance
```

`DisableDefaultEC2SecurityGroup` および `BlockDeviceMappings`は コンソールでは使用できません。

`BlockDeviceMappings` では、インスタンス用の追加のブロックデバイスを設定できます。詳細については、「Amazon EC2 ユーザーガイド」の「ブロックデバイスマッピング」を参照してください。

EB CLI および Elastic Beanstalk コンソールでは、上記のオプションに推奨値が適用されます。設定ファイルを使用して同じファイルを設定する場合は、これらの設定を削除する必要があります。詳細については、「推奨値」を参照してください。

Elastic Beanstalk 環境のインスタンスでの IMDS の設定

このトピックでは、インスタンスメタデータサービス (IMDS) について説明します。

インスタンスメタデータは、アプリケーションが実行中のインスタンスを設定または管理するために使用できる Amazon Elastic Compute Cloud (Amazon EC2) インスタンスに関連するデータです。インスタンスメタデータサービス (IMDS) は、インスタンス上のコードによって、インスタンスメタデータに安全にアクセスするために使用されるインスタンス上のコンポーネントです。このコード

は、環境インスタンスの Elastic Beanstalk プラットフォームコード、アプリケーションが使用している AWS SDK、またはアプリケーション独自のコードです。詳細については「アマゾン EC2 ユーザーガイド」の「Instance metadata and user data」(インスタンスメタデータとユーザーデータ) を参照してください。

コードは、手法としてインスタンスメタデータサービスバージョン 1 (IMDSv1) またはインスタンスメタデータサービスバージョン 2 (IMDSv2) のいずれかを使用して、実行中のインスタンスからインスタンスメタデータにアクセスできます。IMDSv2 はセッション指向のリクエストを使用し、IMDS へのアクセス試行に利用される可能性があるいくつかのタイプの脆弱性を軽減します。これら 2 つの方法については、「Amazon EC2 ユーザーガイド」のインスタンスメタデータサービスの設定に関するページを参照してください。

トピック

- IMDSのプラットフォームサポート
- IMDS 手法の選択
- Elastic Beanstalk コンソールを使用した IMDS の設定
- aws:autoscaling:launchconfiguration 名前空間

IMDSのプラットフォームサポート

Amazon Linux 2、Amazon Linux 2023、および Windows サーバーで実行されている Elastic Beanstalk プラットフォームはすべて IMDSv1 と IMDSv2 の両方をサポートしています。詳細については、Elastic Beanstalk コンソールを使用した IMDS の設定を参照してください。

IMDS 手法の選択

ご使用の環境でサポートする IMDS 手法を決定する際には、次のユースケースについて考慮してください。

- AWS SDK – アプリケーションが AWS SDK を使用している場合は、必ず最新バージョンの SDK を使用してください。 AWS SDKs IMDS 呼び出しを行い、新しい SDK バージョンは可能な限り IMDSv2 を使用します。IMDSv1 を無効にすると、アプリケーションで以前の SDK バージョンが使用されている場合に、IMDS 呼び出しが失敗することがあります。
- アプリケーションコード – アプリケーションが IMDS 呼び出しを行う場合は、直接 HTTP リクエストを行う代わりに呼び出しを行うことができるように AWS SDK の使用を検討してください。このようにすると、異なる IMDS 手法間で切り替えるためにコードを変更する必要はありません。 AWS SDK は可能な限り IMDSv2 を使用します。

- Elastic Beanstalk プラットフォームコード – このコードは AWS SDK を介して IMDS 呼び出しを行うため、サポートされているすべてのプラットフォームバージョンで IMDSv2 を使用します。コードがup-to-date AWS SDK を使用していて、SDK を介してすべての IMDS 呼び出しを行う場合は、IMDSv1 を安全に無効にできます。

Elastic Beanstalk コンソールを使用した IMDS の設定

Elastic Beanstalk コンソールで、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を変更できます。

> **Important**
>
> `DisableIMDSv1` オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してください Elastic Beanstalk 環境を起動テンプレートに移行する 。

Elastic Beanstalk コンソールで Amazon EC2 インスタンスの IMDS を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [Instance traffic and scaling] (インスタンスのトラフィックおよびスケーリング) 設定カテゴリで、[Edit] (編集) を選択します。
5. IMDSv2 を適用するには、[Disable IMDSv1 (IMDSv1 の無効化)] を設定します。IMDSv1 と IMDSv2 の両方を有効にするには、[Disable IMDSv1 (IMDSv1 の無効化)] をオフにします。
6. ページの最下部で [適用] を選択し変更を保存します。

aws:autoscaling:launchconfiguration 名前空間

`aws:autoscaling:launchconfiguration` 名前空間の設定オプションを使用して、環境のインスタンスで IMDS を設定します。

> **Important**
>
> `DisableIMDSv1` オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してください Elastic Beanstalk 環境を起動テンプレートに移行する 。

次に示す設定ファイル例では、`DisableIMDSv1` オプションを使用して IMDSv1 を無効にしています。

```
option_settings:
  aws:autoscaling:launchconfiguration:
    DisableIMDSv1: true
```

IMDSv1 を無効にして IMDSv2 が適用されるようにするには、DisableIMDSv1 を `true` に設定します。

IMDSv1 と IMDSv2 の両方を有効にするには、DisableIMDSv1 を `false` に設定します。

Elastic Beanstalk 環境インスタンスの Auto Scaling

このトピックでは、Auto Scaling 機能をカスタマイズして Elastic Beanstalk 環境のワークロードを管理する方法について説明します。環境の Auto Scaling は、Elastic Beanstalk コンソール、名前空間設定オプション、、AWS CLIまたは EB CLI を使用して設定できます。

負荷分散されたインスタンス環境または単一インスタンス環境

AWS Elastic Beanstalk 環境には、環境内の Amazon EC2 インスタンスを管理する Auto Scaling グループが含まれています。単一インスタンス環境では、Auto Scaling グループは常に1 つのインスタンスが実行されているよう確認します。負荷分散される環境では、実行する範囲のグループを設定すると、Auto Scaling は、負荷に基づき、必要に応じてインスタンスを追加または削除します。

EC2 インスタンス設定

Auto Scaling グループは、設定の選択を適用して、環境内の EC2 インスタンスをプロビジョニングおよび管理します。EC2 設定を変更して、インスタンスタイプ、キーペア、Amazon Elastic Block

Store (Amazon EBS) ストレージ、およびインスタンスの起動時にのみ設定できるその他の設定を変更できます。

オンデマンドインスタンスとスポットインスタンス

オプションとして、Elastic Beanstalk は環境にスポットインスタンスを含め、オンデマンドインスタンスと組み合わせて管理できます。スポットインスタンスの可用性に影響する変更をモニタリングし、自動的に応答するように Amazon EC2 Auto Scaling を設定するには、容量の再調整を有効にします。Auto Scaling サービススポット配分戦略が環境にスポットインスタンスをプロビジョニングするために使用する を設定することもできます。

スポットインスタンスを有効にするときに必要なアクセス許可

スポットインスタンスリクエストを有効にするには、Amazon EC2 起動テンプレートを使用する必要があります。この機能を環境の作成時や更新時に設定すると、Elastic Beanstalk は Amazon EC2 起動テンプレートを使用するように環境を設定しようとします (環境でまだ使用されていない場合)。この場合、ユーザーポリシーに必要なアクセス許可がないと、環境の作成や更新は失敗する可能性があります。したがって、管理ユーザーポリシーを使用するか、カスタムポリシーに必要なアクセス許可を追加することをお勧めします。必要なアクセス許可の詳細については、「起動テンプレートに必要なアクセス許可」を参照してください。

Auto Scaling トリガー

Auto Scaling グループは、2 つの Amazon CloudWatch アラームを使用してスケーリングオペレーションをトリガーします。各インスタンスの 5 分間の平均アウトバウンドネットワークトラフィックが 6 MiB 以上または 2 MiB 以下の場合は、デフォルトのトリガーがスケーリングされます。Auto Scaling を効率的に使用するには、アプリケーション、インスタンスタイプ、サービス要件に合ったトリガーを設定します。レイテンシー、ディスク I/O、CPU 使用率、リクエスト数などの複数の統計に基づいて、スケールすることができます。

Auto Scaling アクションをスケジュールする

ピークトラフィックが予測される期間に、環境での Amazon EC2 インスタンスの使用を最適化する場合は、Auto Scaling グループを設定し、スケジュールに基づいてインスタンス数を変更します。毎日または週 1 回繰り返すグループの設定への変更、またはワンタイムへの変更をスケジュールして、多量のトラフィックをサイトへ誘導するマーケティングイベント用に準備することができます。

Auto Scaling ヘルスチェック

Auto Scaling は、起動した各 Amazon EC2 インスタンスの状態をモニタリングします。インスタンスが予期せずに終了した場合、Auto Scaling は終了を検出し、代わりのインスタンスを起動します。

グループを設定して、ロードバランサーのヘルスチェックメカニズムを使用する方法については、「Elastic Beanstalk 環境の Auto Scaling ヘルスチェック設定」を参照してください。

トピック

- Elastic Beanstalk 環境を起動テンプレートに移行する
- Elastic Beanstalk 環境のスポットインスタンスのサポート
- Elastic Beanstalk 環境の Auto Scaling トリガー
- Elastic Beanstalk 環境のスケジュールされた Auto Scaling アクション
- Elastic Beanstalk 環境の Auto Scaling ヘルスチェック設定

Elastic Beanstalk 環境を起動テンプレートに移行する

2024 年 10 月 1 日現在、Amazon EC2 Auto Scaling は新しいアカウントの起動設定をサポートしていません。その日付より前に作成されたアカウントには、起動設定がある場合があります。

以下の利点のために、起動テンプレートに移行することをお勧めします。

- アプリケーションの可用性の向上
- Auto Scaling グループのワークロードの最適化の向上
- 最新の EC2 および Auto Scaling 機能へのアクセス

詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Auto Scaling 起動設定」を参照してください。

起動テンプレートのオプション設定

環境を起動設定から起動テンプレートに移行するには、次のいずれかの設定オプションを設定します。

- `RootVolumeType` オプションを gp3 に設定。このオプションは、コンソールまたは名前空間を使用して設定できます。
- `BlockDeviceMappings` オプションには gp3 が含まれます。このオプションは、コンソールまたは名前空間を使用して設定できます。
- `DisableIMDSv1` オプションを true に設定。このオプションは、名前空間を使用して設定することをお勧めします。

- `EnableSpot` オプションを true に設定。詳細については、「スポットインスタンスの有効化」を参照してください。

> **Important**
> 環境が起動テンプレートの使用を開始した後、起動テンプレートの使用を最初にトリガーした設定オプションを削除しても、Elastic Beanstalk は起動設定に戻しません。

環境に起動設定があるか起動テンプレートがあるかを確認します。

CloudFormation スタックテンプレートを調べることで、環境が起動テンプレートを既に使用しているか、起動設定を使用しているかを確認できます。

環境の CloudFormation スタックテンプレートを調べるには

1. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開きます。
2. 画面上部のナビゲーションバーで、環境を作成した AWS リージョンを選択します。
3. CloudFormation コンソールの [スタック] ページで、[説明] 列を調べます。

   Elastic Beanstalk 環境のスタックを見つけて選択します。CloudFormation により、環境のスタックの詳細が表示されます。
4. [スタックの詳細] で [テンプレート] タブを選択します。

   ブラウザのページ検索を使用して、テンプレートテキストで launchtemplate または launchconfiguration を検索できます。

詳細については、「AWS CloudFormation ユーザーガイド」の「スタック情報の表示」を参照してください。

起動テンプレートに必要なアクセス許可

デフォルトの Elastic Beanstalk マネージドサービスロールポリシー AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy は、起動テンプレートを作成および管理するために必要なアクセス許可を提供します。Elastic Beanstalk は、環境の作成など、多くの環境オペレーションを完了するには、起動テンプレートを管理する必要があります。

カスタムポリシーを Elastic Beanstalk サービスロールにアタッチする場合は、サービスロールに起動テンプレートを作成するための次のアクセス許可が含まれていることを確認します。これらのアクセス許可により、Elastic Beanstalk はアカウントで環境を正常に作成および更新できます。

Amazon EC2 起動テンプレートに必要なアクセス許可

- `ec2:RunInstances`
- `ec2:CreateLaunchTemplate`
- `ec2:CreateLaunchTemplateVersions`
- `ec2:DeleteLaunchTemplate`
- `ec2:DeleteLaunchTemplateVersions`
- `ec2:DescribeLaunchTemplate`
- `ec2:DescribeLaunchTemplateVersions`

次の IAM ポリシーの例には、これらのアクセス許可が含まれています。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:RunInstances",
        "ec2:CreateLaunchTemplate",
        "ec2:CreateLaunchTemplateVersions",
        "ec2:DeleteLaunchTemplate",
        "ec2:DeleteLaunchTemplateVersions",
        "ec2:DescribeLaunchTemplate",
        "ec2:DescribeLaunchTemplateVersions"
      ],
      "Resource": [
                "*"
            ]
     }
  ]
}
```

詳細については、「Elastic Beanstalk サービスロールの管理」および「Elastic Beanstalk ユーザーポリシーの管理」を参照してください。

起動テンプレートの詳細

起動テンプレートの詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「Auto Scaling 起動テンプレート」を参照してください。

起動テンプレート AWS への移行とそれらが提供する利点の詳細については、AWS 「 コンピューティングブログ」のAmazon EC2 Auto Scaling は起動設定に新しい EC2 機能のサポートを追加しません」を参照してください。

> **Important**
>
> このブログ記事で参照されている手順に従って、古い環境を起動テンプレートに移行する必要はありません。既存の Elastic Beanstalk 環境を起動テンプレートに移行するには、 にリストされているオプションのいずれかを設定します起動テンプレートのオプション設定。

Elastic Beanstalk 環境のスポットインスタンスのサポート

このトピックでは、Elastic Beanstalk 環境でスポットインスタンスの容量と負荷分散を管理できる設定オプションについて説明します。また、これらのオプションの設定に使用できるメソッドの詳細と例も示します。Elastic Beanstalk コンソール、名前空間設定オプション、AWS CLI、または EB CLI を使用して、設定オプションを管理できます。

容量の再調整によるスポットインスタンスの中断の最小化

アプリケーションへのスポットインスタンスの中断の影響を最小限に抑えるために、Amazon EC2 Auto Scaling に含まれている容量の再調整オプションを有効にできます。

> **Important**
>
> スポットインスタンスに対する需要は刻一刻と大幅に変化する可能性があります。また、スポットインスタンスの可用性も利用可能な未使用の Amazon EC2 インスタンスの数に応じて大きく変化する可能性があります。スポットインスタンスが中断される可能性は常にあります。

キャパシティの再調整を有効にすると、EC2 は Auto Scaling グループのスポットインスタンスが中断される前に自動的に置き換えようとします。この機能を有効にするには、Elastic Beanstalk コンソールを使用して、Auto Scaling グループを設定します。または、Elastic Beanstalk

`EnableCapacityRebalancing` 設定オプションを aws:autoscaling:asg 名前空間の `true` に設定することもできます。

詳細については、「Amazon EC2 Auto Scaling ユーザーガイド」の「容量の再調整」、および「Amazon EC2 ユーザーガイド」の「スポットインスタンスの中断」を参照してください。

古いインスタンスタイプとスポットインスタンスのサポート

一部の古い AWS アカウントでは、スポットインスタンスをサポートしないデフォルトのインスタンスタイプが Elastic Beanstalk に提供される場合があります。スポットインスタンスリクエストを有効にし、エラーが表示された場合は、指定したインスタンスタイプのいずれもスポットをサポートしていません。スポットインスタンスをサポートするインスタンスタイプで設定を更新します。スポットインスタンスタイプを選択するには、スポットインスタンスアドバイザーを使用します。

トピック

- 環境のスポットインスタンスの有効化
- スポットインスタンス配分戦略
- オンデマンドインスタンスとスポットインスタンスの管理
- Elastic Beanstalk 環境の容量設定

環境のスポットインスタンスの有効化

Amazon EC2 スポットインスタンスを利用するには、環境の `EnableSpot`オプションを設定します。環境の Auto Scaling グループは、Amazon EC2 購入オプションを組み合わせて、オンデマンドインスタンスとスポットインスタンスの組み合わせを維持します。

Elastic Beanstalk コンソール、名前空間設定オプション、AWS CLI、または EB CLI を使用して、環境のスポットインスタンスリクエストを有効にできます。

環境でスポットインスタンスを有効にする前に、使用可能な Auto Scaling、容量、ロードバランシングの設定オプションについて理解しておく必要があります。ワークロード、インスタンスの中断の影響、および料金に関連するアプリケーションの要件は、スポットインスタンスを有効にする計画における重要な考慮事項です。

以下のトピックでは、Auto Scaling とキャパシティ管理オプションの詳細と、それらの組み合わせによる環境への影響について説明します。さまざまなオプションとその設定方法について通知およびガイドする手順と設定例があります。また、設定の管理やイベントへの対応に役立つツールや機能も提供しています。トラフィックの予測可能な期間に基づいて設定の自動変更をスケジュールしたり、

トラフィック量などの要因に対応するトリガーを設定したり、Auto Scaling のモニタリングとヘルスチェックを設定したりできます。

主要な概念やベストプラクティスの説明など、スポットインスタンスの詳細については、Amazon EC2 ユーザーガイド」の「スポットインスタンス」を参照してください。

> **Important**
>
> `EnableSpot` オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してくださいElastic Beanstalk 環境を起動テンプレートに移行する 。

スポットインスタンス配分戦略

Elastic Beanstalk 環境については、このトピックに記載されている配分戦略のいずれかを選択できます。Elastic Beanstalk コンソール、名前空間設定オプション、または を使用してAWS CLI、環境のスポットインスタンス割り当て戦略および関連属性を設定および設定します。

Amazon EC2 は、配分戦略を適用して、環境のスポットインスタンスを管理およびプロビジョニングします。各配分戦略は、使用可能な容量、料金、インスタンスタイプの選択を処理するように定義されている方法に基づいて、割り当てられたインスタンスを最適化します。

Amazon EC2 Auto Scaling は、スポットインスタンスに次の配分戦略を提供します。

- 容量最適化 (デフォルト)
  - 起動するインスタンスの数に最適な容量で、プールからスポットインスタンスをリクエストします。
  - この戦略は、サービスの中断の可能性を最小限に抑える必要があるワークロードに適しています。
- 料金キャパシティ最適化
  - 中断の可能性が最も低く、価格が最も低いプールからスポットインスタンスをリクエストします。
  - これは、ほとんどのスポットワークロードに推奨される選択肢です。
- キャパシティ最適化の優先順位

  - ベストエフォートベースで選択したインスタンスタイプの優先順位付けを尊重しながら、最初に容量の可用性に基づいてスポットインスタンスをリクエストします。 Elastic Beanstalk のスポットインスタンスオプションを設定するときに、優先度順にインスタンスタイプのリストを指定できます。
  - この戦略は、サービスの中断を最小限に抑え、特定のインスタンスタイプの優先順位付けが重要なワークロードに適しています。
- 最低料金
  - 利用可能なインスタンスを持つ最低価格のプールからスポットインスタンスをリクエストします。
  - この戦略を使用するときは注意することが重要です。これは、インスタンス料金のみを考慮し、容量の可用性は考慮しないため、中断率が高くなるためです。

各配分戦略の詳細については、Amazon EC2 Auto Scaling ユーザーガイド」の「複数のインスタンスタイプの配分戦略」を参照してください。

環境の要件を満たすのに最適な配分戦略を理解するには、Amazon EC2 ユーザーガイド」の「適切なスポット配分戦略を選択する」を参照してください。

オンデマンドインスタンスとスポットインスタンスの管理

1 つの Auto Scaling グループ内で、オンデマンドインスタンスとスポットインスタンスのフリートを起動してオートスケールできます。以下のオプションを並行して使用して、Auto Scaling サービスが環境内のスポットインスタンスとオンデマンドインスタンスを管理する方法を設定できます。

これらのオプションは、Elastic Beanstalk コンソール、名前空間設定オプション、、AWS CLIまたは EB CLI を使用して環境に設定できます。

これらのオプションは、aws:ec2:instances 名前空間の一部です。

- `EnableSpot` - `true`この設定に設定すると、環境のスポットインスタンスリクエストが有効になります。
- `SpotFleetOnDemandBase` - 環境のスケールアップ時にスポットインスタンスを考慮する前に、Auto Scaling グループがプロビジョニングするオンデマンドインスタンスの最小数を設定します。
- `SpotFleetOnDemandAboveBasePercentage` - Auto Scaling グループがインスタンスを超えてプロビジョニングする追加の容量の一部としてのオンデマンド`SpotOnDemandBase`インスタンスの割合。

前述のオプションは、aws:autoscaling:asg 名前空間の以下のオプションと相関しています。

- `MinSize` - Auto Scaling グループに含めるインスタンスの最小数。
- `MaxSize` - Auto Scaling グループに含めるインスタンスの最大数。

> **Important**
>
> `EnableSpot` オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してください Elastic Beanstalk 環境を起動テンプレートに移行する 。

両方の名前空間オプションのセットの適用

以下のポイントは、これらのオプション設定の組み合わせが環境のスケーリングにどのように影響するかを説明しています。

- `MinSize` のみが、環境の初期容量 (最小で実行するインスタンスの数) を決定します。
- `SpotFleetOnDemandBase` は初期容量には影響しません。スポットが有効になっている場合、このオプションは、スポットインスタンスが考慮される前にプロビジョニングされるオンデマンドインスタンスの数を決定します。
- `SpotFleetOnDemandBase` が `MinSize` よりも小さい場合を考慮します。それでも、初期容量どおりの `MinSize` インスタンスが得られます。そのうち少なくとも `SpotFleetOnDemandBase` はオンデマンドインスタンスである必要があります。
- `SpotFleetOnDemandBase` が `MinSize` よりも大きい場合を考慮します。環境のスケールアウト時には、2 つの値の差に等しい数のインスタンスが追加されることが保証されます。つまり、`SpotFleetOnDemandBase` 要件を満たす前に、少なくともオンデマンドである追加の (`SpotFleetOnDemandBase - MinSize`) インスタンスを取得することが保証されています。

単一インスタンス環境

本稼働環境では、スポットインスタンスは、スケーラブルな負荷分散された環境の一部として特に便利です。単一インスタンス環境でスポットを使用することはお勧めしません。スポットインスタンスが使用できない場合、環境の容量全体 (単一インスタンス) が失われる可能

性があります。開発またはテストを目的として、単一インスタンス環境でのスポットインスタンスの使用を希望する場合があるとします。その場合は、SpotFleetOnDemandBase と SpotFleetOnDemandAboveBasePercentage の両方を必ずゼロに設定してください。その他の設定を行うと、オンデマンドインスタンスになります。

スケーリングオプション設定の例

次の例は、幅広いスケーリングオプションを設定するさまざまなシナリオを示しています。すべての例において、スポットインスタンスリクエストを有効にした負荷分散環境が想定されています。

Example 1: 初期容量の一部としてのオンデマンドとスポット

オプション設定

オプション	名前空間	値
MinSize	aws:autoscaling:asg	10
MaxSize	aws:autoscaling:asg	24
SpotFleetOnDemandBase	aws:ec2:instances	4
SpotFleetOnDemandAboveBasePercentage	aws:ec2:instances	50

この例では、環境は 10 個のインスタンスから始まります。そのうちの 7 個はオンデマンド (4 つのベース、上記 6 つのベースの 50%) で、3 個はスポットです。環境はインスタンスを最大 24 個にスケールアウトできます。スケールアウトすると、4 つのベースオンデマンドインスタンスの上にあるフリートの部分のオンデマンド部分は 50% に保たれ、全体では最大 24 のインスタンスが維持されます。そのうちの 14 はオンデマンド (4 つのベース、上記 20 のうち 50%) 、10 はスポットです。

Example 2: すべてのオンデマンドの初期容量

オプション設定

オプション	名前空間	値
MinSize	aws:autoscaling:asg	4

オプション	名前空間	値
MaxSize	aws:autoscaling:asg	24
SpotFleetOnDemandBase	aws:ec2:instances	4
SpotFleetOnDemandAboveBasePercentage	aws:ec2:instances	50

この例では、環境は 4 つのインスタンスから始まり、そのすべてがオンデマンドです。環境はインスタンスを最大 24 個にスケールアウトできます。スケールアウトすると、4 つのベースオンデマンドインスタンスの上にあるフリートの部分のオンデマンド部分は 50% に保たれ、全体では最大 24 のインスタンスが維持されます。そのうちの 14 はオンデマンド (4 つのベース、上記 20 のうち 50%) 、10 はスポットです。

Example 3: 初期容量を超える追加のオンデマンドベース

オプション設定

オプション	名前空間	値
MinSize	aws:autoscaling:asg	3
MaxSize	aws:autoscaling:asg	24
SpotFleetOnDemandBase	aws:ec2:instances	4
SpotFleetOnDemandAboveBasePercentage	aws:ec2:instances	50

この例では、環境は 3 つのインスタンスから始まり、そのすべてがオンデマンドです。環境はインスタンスを最大 24 個にスケールアウトできます。初期の 3 つ上の最初の追加インスタンスはオンデマンドで、4 つのベースオンデマンドインスタンスを完成させます。さらにスケールアウトすると、4 つのベースオンデマンドインスタンスの上にあるフリートの部分のオンデマンド部分は 50% に保たれ、全体では最大 24 のインスタンスが維持されます。そのうちの 14 はオンデマンド (4 つのベース、上記 20 のうち 50%) 、10 はスポットです。

Elastic Beanstalk 環境の容量設定

このトピックでは、Elastic Beanstalk 環境の Auto Scaling 容量を設定するさまざまなアプローチについて説明します。Elastic Beanstalk コンソール、EB CLI、 AWS CLI、または名前空間オプションを使用できます。

> **Important**
>
> `EnableSpot` オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してください Elastic Beanstalk 環境を起動テンプレートに移行する 。

コンソールを使用した設定

Elastic Beanstalk コンソールの環境の設定ページでキャパシティを編集することで、Auto Scaling グループのキャパシティ管理を設定できます。

Elastic Beanstalk コンソールで Auto Scaling グループの容量を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [キャパシティー] 設定カテゴリで、[編集] を選択します。
5. [Auto Scaling グループ] セクションで、次のように設定します。
   - [環境タイプ] – [負荷分散] を選択します。
   - [最小インスタンス数] – グループに一度に含まれる EC2 インスタンスの最小数。グループは最小数から開始し、スケールアップのトリガー条件が満たされるとインスタンスが追加されます。
   - [最大インスタンス数] – グループに一度に含まれる EC2 インスタンスの最大数。

> **Note**
>
> ローリング更新を使用する場合、ローリング更新の最小インスタンス数が [サービス内のインスタンスの最小数] の設定よりも多いことを確認します。

- フリート組成 — デフォルトはオンデマンドインスタンスです。スポットインスタンス要件を有効にするには、[Combined purchase options and instance (購入オプションとインスタンスの組み合わせ)] を選択します。

> **Important**
>
> `EnableSpot` オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してくださいElastic Beanstalk 環境を起動テンプレートに移行する 。

スポットインスタンス要件を有効にすることを選択すると、次のオプションが有効になります:

- スポット配分戦略 – 使用可能な容量、料金、インスタンスタイプの選択に基づいて、環境でスポットインスタンスを管理およびプロビジョニングするために使用される方法を決定します。容量最適化 (デフォルト) 、価格容量最適化、容量最適化優先、または最低価格から選択します。各配分戦略の説明と詳細については、「」を参照してくださいthe section called “スポット配分戦略”。
- 上限スポット価格 – スポットインスタンスの上限価格オプションに関する推奨事項については、「Amazon EC2 ユーザーガイド」の「スポットインスタンスの料金履歴」を参照してください。
- オンデマンドベース - 環境のスケールアウト時にスポットインスタンスを考慮する前に、Auto Scaling グループがプロビジョニングするオンデマンドインスタンスの最小数。
- ベースを超えるオンデマンド - Auto Scaling グループがオンデマンドベースのインスタンスを超えてプロビジョニングする追加容量の一部としてのオンデマンドインスタンスの割合。

> **Note**
>
> オンデマンドベースおよびベースを超えるオンデマンドは、前述の最小および最大インスタンスオプションと関係しています。これらのオプションと例の詳細については、「the section called “スポットインスタンスのサポート”」を参照してください。

- キャパシティの再調整 – このオプションは、Auto Scaling グループに少なくとも 1 つのスポットインスタンスがある場合にのみ関連します。この機能を有効にすると、EC2 は Auto Scaling グループのスポットインスタンスが中断される前に自動的に置き換えを試み、アプリケーションのスポットインスタンスの中断を最小限に抑えます。詳細については、Amazon EC2 Auto Scaling ユーザーガイド」の「キャパシティの再調整」を参照してください。

- アーキテクチャ – EC2 インスタンスのプロセッサアーキテクチャ。プロセッサアーキテクチャは、次のフィールドで使用できる EC2 インスタンスタイプを決定します。
- インスタンスタイプ – アプリケーションを実行するために起動された Amazon EC2 インスタンスのタイプ。詳細については、「the section called “インスタンスのタイプ”」を参照してください。
- AMI ID – Elastic Beanstalk が環境で Amazon EC2 インスタンスを起動するために使用するマシンイメージ。詳細については、「the section called “AMI ID”」を参照してください。
- [アベイラビリティーゾーン] – アベイラビリティーゾーンの数を選択し、環境のインスタンスを分散させます。デフォルトでは、Auto Scaling グループは、使用可能なゾーン間で均等にインスタンスを起動します。少ないゾーンにインスタンスを集中させるには、使用するゾーンの数を選択します。実稼働環境では、2 つ以上のゾーンを使用して、一方のアベイラビリティーゾーンが使用できなくなってもアプリケーションが利用できるようにします。
- [配置] (オプション) – 使用するアベイラビリティーゾーンを選択します。インスタンスを特定のゾーンのリソースに接続する必要がある場合、またはゾーン固有のリザーブドインスタンスを購入済みの場合は、この設定を使用します。カスタム VPC で環境を起動する場合は、このオプションを設定できません。カスタム VPC では、環境に割り当てるサブネットのアベイラビリティーゾーンを選択します。
- [スケーリングクールダウン] – スケーリング後、トリガーの評価を継続する前に、インスタンスが起動または終了するまでの時間 (秒)。詳細については、「スケーリングクールダウン」を参照してください。

6. ページの最下部で [適用] を選択し変更を保存します。

名前空間オプションを使用した設定

Elastic Beanstalk には、aws:autoscaling:asg および aws:ec2:instances という 2 つの名前空間で、Auto Scaling 設定の設定オプションが用意されています。

aws:autoscaling:asg 名前空間

aws:autoscaling:asg ??? 名前空間は、全体的なスケールと可用性のオプションを提供します。

次の設定ファイルの例では、2～4 個のインスタンス、特定のアベイラビリティーゾーン、および 12 分間 (720 秒間) のクールダウン期間を使用するように Auto Scaling グループを設定します。これにより、スポットインスタンスの容量の再調整が有効になります。このオプションは、次の設定ファイルの例に示すように、aws:ec2:instances名前空間trueで が EnableSpotに設定されているEnableCapacityRebalancing場合にのみ有効になります。

```
option_settings:
  aws:autoscaling:asg:
    Availability Zones: Any
    Cooldown: '720'
    Custom Availability Zones: 'us-west-2a,us-west-2b'
    MaxSize: '4'
    MinSize: '2'
    EnableCapacityRebalancing: true
```

aws:ec2:instances 名前空間

> **Note**
>
> 環境設定を更新し、InstanceTypes オプションから 1 つ以上のインスタンスタイプを削除すると、Elastic Beanstalk は、削除されたインスタンスタイプで実行されているすべての Amazon EC2 インスタンスを終了します。次に、環境の Auto Scaling グループは、現在指定されているインスタンスタイプを使用して、必要に応じて新しいインスタンスを起動し、必要な容量を完了します。

aws:ec2:instances 名前空間は、スポットインスタンスの管理など、環境のインスタンスに関連するオプションを提供します。aws:autoscaling:launchconfiguration と aws:autoscaling:asg を補完します。

次の設定ファイルの例では、環境のスポットインスタンス・リクエストを有効にするように Auto Scaling グループを設定します。使用できる 3 つのインスタンスタイプを指定します。ベースライン

容量として少なくとも 1 つのオンデマンドインスタンスが使用され、任意の追加容量として、持続する 33% のオンデマンドインスタンスが使用されます。

設定では、スポット配分戦略を に設定しますcapacity-optimized-prioritized。この特定の配分戦略では、 InstanceTypesオプションで指定されたインスタンスタイプの順序に基づいて、プールからのインスタンス起動に優先順位を付けます。が指定されていない場合SpotAllocationStrategy、デフォルトで になりますcapacity-optimized。

```
option_settings:
  aws:ec2:instances:
    EnableSpot: true
    InstanceTypes: 't2.micro,t3.micro,t3.small'
    SpotAllocationStrategy: capacity-optimized-prioritized
    SpotFleetOnDemandBase: '1'
    SpotFleetOnDemandAboveBasePercentage: '33'
```

スポットインスタンスタイプを選択するには、スポットインスタンスアドバイザーを使用します。

> **Important**
>
> EnableSpot オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してくださいElastic Beanstalk 環境を起動テンプレートに移行する 。

を使用した設定 AWS CLI

このセクションでは、 AWS CLI create-environment コマンドを使用して、これらのセクションで説明されている Auto Scaling オプションとキャパシティオプションを使用して環境を設定する方法の例を示します。前の名前空間設定オプションセクションで説明したようにaws:ec2:instances、aws:autoscaling:asgと の名前空間設定もこの例で設定されています。 ???

AWS コマンドラインインターフェイスには、Elastic Beanstalk 環境を作成および設定するためのコマンドが用意されています。--option-settings オプションを使用すると、Elastic Beanstalk でサポートされている名前空間オプションを渡すことができます。つまり、前述の名前空間設定オプションを該当する AWS CLI コマンドに渡して、Elastic Beanstalk 環境を設定できます。

> **Note**
>
> で update-environment コマンドを使用して`--option-settings`、名前空間オプションを追加または更新することもできます。環境から名前空間オプションを削除する必要がある場合は、 で update-environment コマンドを使用します`--options-to-remove`。

次の の例では、新しい環境を作成します。渡されるオプションの詳細については、前のトピックの名前空間設定オプションを参照してください。

aws:autoscaling:launchconfiguration 名前空間`IamInstanceProfile`にリストされている fist オプションは、Elastic Beanstalk インスタンスプロファイルです。これは、新しい環境を作成するときに必要です。

Example — Auto Scaling オプションを使用した環境の作成 (名前空間オプションをインライン)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v4.3.0 running Python 3.12" \
--option-settings \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-elasticbeanstalk-ec2-role
Namespace=aws:autoscaling:asg,OptionName=Availability Zones,Value=Any \
Namespace=aws:autoscaling:asg,OptionName=Cooldown,Value=720 \
Namespace=aws:autoscaling:asg,OptionName=Custom Availability Zones,Value=us-west-2a,us-west-2b \
Namespace=aws:autoscaling:asg,OptionName=MaxSize,Value=4 \
Namespace=aws:autoscaling:asg,OptionName=MinSize,Value=2 \
Namespace=aws:autoscaling:asg,OptionName=EnableCapacityRebalancing,Value=true \
Namespace=aws:ec2:instances,OptionName=EnableSpot,Value=true \
Namespace=aws:ec2:instances,OptionName=InstanceTypes,Value=t2.micro,t3.micro,t3.small \
Namespace=aws:ec2:instances,OptionName=SpotAllocationStrategy,Value=capacity-optimized-prioritized \
Namespace=aws:ec2:instances,OptionName=SpotFleetOnDemandBase,Value=1 \
Namespace=aws:ec2:instances,OptionName=SpotFleetOnDemandAboveBasePercentage,Value=33
```

> **Important**
>
> `EnableSpot` オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してください Elastic Beanstalk 環境を起動テンプレートに移行する 。

別の方法として、`options.json`ファイルを使用して、インラインで名前空間オプションを含めるのではなく、名前空間オプションを指定します。

Example —Auto Scaling オプションを使用した環境の作成 (**`options.json`** ファイルの名前空間オプション)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v4.3.0 running Python 3.12"
--option-settings file://options.json
```

Example

```
### example options.json ###
[
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "IamInstanceProfile",
        "Value": "aws-elasticbeanstalk-ec2-role"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "Availability Zones",
        "Value": "Any"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "Cooldown",
```

```
        "Value": "720"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "Custom Availability Zones",
        "Value": "us-west-2a,us-west-2b"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MaxSize",
        "Value": "4"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MinSize",
        "Value": "2"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "EnableCapacityRebalancing",
        "Value": "true"
    },
    {
        "Namespace": "aws:ec2:instances",
        "OptionName": "EnableSpot",
        "Value": "true"
    },
    {
        "Namespace": "aws:ec2:instances",
        "OptionName": "InstanceTypes",
        "Value": "t2.micro,t3.micro,t3.small"
    },
    {
        "Namespace": "aws:ec2:instances",
        "OptionName": "SpotAllocationStrategy",
        "Value": "capacity-optimized-prioritized"
    },
    {
        "Namespace": "aws:ec2:instances",
        "OptionName": "SpotFleetOnDemandBase",
        "Value": "1"
    },
    {
        "Namespace": "aws:ec2:instances",
```

```
        "OptionName": "SpotFleetOnDemandAboveBasePercentage",
        "Value": "33"
    }
]
```

EB CLI を使用した設定

eb create コマンドを使用して環境を作成する場合、環境の Auto Scaling グループに関連するいくつかのオプションを指定できます。これは、環境の容量を制御するのに役立つオプションの一部です。

`--single`

1 つの Amazon EC2 インスタンスが含まれ、ロードバランサーがない環境を作成します。このオプションを使用しない場合、作成された環境にロードバランサーが追加されます。

`--enable-spot`

環境のスポットインスタンスリクエストを有効にします。

> **Important**
>
> `enable-spot` オプション設定により、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してくださいElastic Beanstalk 環境を起動テンプレートに移行する。

eb create コマンドの以下のオプションは、`--enable-spot` でのみ使用できます。

`--instance-types`

環境で使用する Amazon EC2 インスタンスタイプを一覧表示します。

`--spot-max-price`

お客様がスポットインスタンスに対して支払ってもよいと考えるユニット時間あたりの上限価格 (米ドル)。スポットインスタンスの上限価格オプションに関する推奨事項については、「Amazon EC2 ユーザーガイド」の「スポットインスタンスの料金履歴」を参照してください。

`--on-demand-base-capacity`

環境のスケールアップ時にスポットインスタンスを考慮する前に、Auto Scaling グループがプロビジョニングするオンデマンドインスタンスの最小数。

`--on-demand-above-base-capacity`

`--on-demand-base-capacity` オプションで指定されたインスタンス数を超えて Auto Scaling グループがプロビジョニングする追加容量の一部としてのオンデマンドインスタンスの割合。

次の例では、環境を作成し、新しい環境のスポットインスタンスリクエストを有効にするように Auto Scaling グループを設定します。この例では、使用可能な 3 つのインスタンスタイプを使用できます。

```
$ eb create --enable-spot --instance-types "t2.micro,t3.micro,t3.small"
```

> **Important**
>
> `--instance-type` (「s」なし) という、同様の名前が付いた別のオプションがあります。EB CLI は、オンデマンドインスタンスの処理時のみ、このオプションを認識します。`--instance-type` (「s」なし) を `--enable-spot` オプションと共に使用しないでください。そうした場合、EB CLI によって無視されます。代わりに、`--instance-types` (「s」あり) を `--enable-spot` オプションと共に使用してください。

Elastic Beanstalk 環境の Auto Scaling トリガー

Elastic Beanstalk 環境の Auto Scaling グループは、2 つの Amazon CloudWatch アラームを使用してスケーリングオペレーションをトリガーします。各インスタンスの 5 分間の平均アウトバウンドネットワークトラフィックが 6 MB 以上または 2 MB 以下の場合は、デフォルトのトリガーがスケーリングされます。Amazon EC2 Auto Scaling を効率的に使用するには、アプリケーション、インスタンスタイプ、サービス要件に合ったトリガーを設定します。レイテンシー、ディスク I/O、CPU 使用率、リクエスト数などの複数の統計に基づいて、スケールすることができます。

CloudWatch メトリクスとアラームの詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch の概念」を参照してください。

Auto Scaling トリガーの設定

Elastic Beanstalk コンソールで、環境の Auto Scaling グループのインスタンス数を調整するトリガーを設定できます。

Elastic Beanstalk コンソールでトリガーを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [キャパシティー] 設定カテゴリで、[編集] を選択します。
5. [スケーリングトリガー] セクションで、次のように設定します。
   - [メトリクス] – Auto Scaling トリガーに使用されるメトリクス。
   - [統計] – `Average` など、トリガーで使用する統計計算。
   - [単位] – トリガーのメトリクス単位 (例: [バイト])。
   - [期間] - ボックスには、Amazon CloudWatch でトリガーのメトリクスを測定する頻度を指定します。
   - [超過時間] – スケーリングオペレーションのトリガーが発生するまでにメトリクスが上限および下限のしきい値を超過する時間 (分単位)。
   - [上限しきい値] – メトリクスが超過時間中にこの数を超えると、スケーリングオペレーションがトリガーされます。
   - [増分をスケールアップする] – 規模の拡大や縮小時に追加する Amazon EC2 インスタンスの数。
   - [下限しきい値] – メトリクスが超過時間中にこの数を下回ると、スケーリングオペレーションがトリガーされます。
   - [増分をスケールダウンする] – 規模の拡大や縮小時に削除する Amazon EC2 インスタンスの数。
6. ページの最下部で [適用] を選択し変更を保存します。

aws:autoscaling:trigger 名前空間

Elastic Beanstalk は、`aws:autoscaling:trigger`名前空間の Auto Scaling 設定に、設定オプションを提供します。この名前空間の設定は、適用されるリソースごとに編成されています。

```
option_settings:
  AWSEBAutoScalingScaleDownPolicy.aws:autoscaling:trigger:
    LowerBreachScaleIncrement: '-1'
  AWSEBAutoScalingScaleUpPolicy.aws:autoscaling:trigger:
    UpperBreachScaleIncrement: '1'
  AWSEBCloudwatchAlarmHigh.aws:autoscaling:trigger:
    UpperThreshold: '6000000'
  AWSEBCloudwatchAlarmLow.aws:autoscaling:trigger:
    BreachDuration: '5'
    EvaluationPeriods: '1'
    LowerThreshold: '2000000'
    MeasureName: NetworkOut
    Period: '5'
    Statistic: Average
    Unit: Bytes
```

Elastic Beanstalk 環境のスケジュールされた Auto Scaling アクション

ピークトラフィックが予測される期間に、環境での Amazon EC2 インスタンスの使用を最適化する場合は、Amazon EC2 Auto Scaling グループを設定し、スケジュールに基づいてインスタンス数を変更します。繰り返しアクションを使用して環境を設定すれば、毎日朝はスケールアップし、トラフィックの低い夜はスケールダウンすることができます。たとえば、マーケティングイベントで一時的にトラフィックをサイトに誘導する場合は、ワンタイムイベントをスケジュールして、開始時にはスケールアップし、終了時にはスケールダウンすることができます。

環境ごとに最大 120 のスケジュールに基づくアクティブアクションを定義することができます。また、Elastic Beanstalk でも、有効期限切れのスケジュールに基づき、最大 150 のアクションを保持し、設定をアップデートして再使用することができます。

スケジュールに基づくアクションの設定

Elastic Beanstalk コンソールで、環境の Auto Scaling グループに対してスケジュールされたアクションを作成できます。

Elastic Beanstalk コンソールでスケジュールされたアクションを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [キャパシティー] 設定カテゴリで、[編集] を選択します。

5. [時間に基づくスケーリング] セクションで、[スケジュールされたアクションの追加] を選択します。
6. 次のスケジュールされたアクション設定を入力します。
   - [名前] – 最大 255 文字の英数字で一意の名前を指定し、スペースは含めないでください。
   - [インスタンス] – Auto Scaling グループに適用する最小および最大インスタンス数を選択します。
   - [希望するキャパシティー] (オプション) – Auto Scaling グループに必要な容量の初期値を設定します。スケジュールに基づくアクションが適用されたら、トリガーは設定に基づき、希望する容量を調整します。
   - [頻度] – [繰り返し] を選択して、スケジュールでスケーリングアクションを繰り返します。
   - [開始時刻] – 1 回限りのアクションの場合は、アクションの実行日時を選択します。

     繰り返しアクションの場合は、開始時間はオプションです。指定するときは、アクションが実行される直近の時間を選択します。この時間の後、アクションは繰り返し式に従って繰り返されます。
   - [Recurrence] – Cron 式を使用して、スケジュールに基づくアクションを実行する頻度を指定します。たとえば、`30 6 * * 2` では、毎週火曜日の午前 6:30 (UTC 時刻) にアクションが実行されます。
   - 終了時間 (オプション) – 繰り返しアクションの場合はオプションです。指定すると、アクションは繰り返し式に従って繰り返され、この時間の後、再度実行されません。

     スケジュールされたアクションが終了しても、Auto Scaling は自動的に以前の設定には戻りません。必要に応じて、Auto Scaling が元の設定に戻るように、スケジュールに基づいて 2 つ目のアクションを設定します。
7. [Add] (追加) を選択します。
8. ページの最下部で [適用] を選択し変更を保存します。

> **Note**
>
> スケジュールされたアクションは適用されるまでは保存されません。

aws:autoscaling:scheduledaction 名前空間

スケジュールに基づくアクションを多数設定する必要がある場合は、設定ファイルまたは Elastic Beanstalk API を使用して、YAML または JSON ファイルから設定オプションの変更を適用することができます。これらの方法では、Suspendオプションにアクセスして、スケジュールに基づく繰り返しアクションを一時的に無効にすることもできます。

> **Note**
>
> スケジュールに基づくアクション設定オプションをコンソール外で操作する場合は、ISO 8601 時刻形式を使用して UTC 時刻で開始時間および終了時間を指定します。たとえば、2015-04-28T04:07:02Z のように指定します。ISO 8601 時刻形式の詳細については、「日付と時刻形式」を参照してください。日付はすべての予定されているアクションで一意である必要があります。

Elastic Beanstalk には、`aws:autoscaling:scheduledaction` 名前空間のスケジュールに基づく設定向けに設定オプションが用意されています。`resource_name` フィールドを使用して、スケジュールに基づくアクションの名前を指定します。

Example Scheduled-scale-up-specific-time-long.config

この設定ファイルでは、2015-12-12T00:00:00Z に 5 インスタンスから 10 インスタンスにスケールアウトするよう Elastic Beanstalk に指示しています。

```
option_settings:
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledScaleUpSpecificTime
    option_name: MinSize
    value: '5'
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledScaleUpSpecificTime
    option_name: MaxSize
    value: '10'
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledScaleUpSpecificTime
    option_name: DesiredCapacity
    value: '5'
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledScaleUpSpecificTime
```

```
    option_name: StartTime
    value: '2015-12-12T00:00:00Z'
```

Example Scheduled-scale-up-specific-time.config

EB CLI または設定ファイルで短縮構文を使用するには、リソース名を名前空間に不可します。

```
option_settings:
  ScheduledScaleUpSpecificTime.aws:autoscaling:scheduledaction:
    MinSize: '5'
    MaxSize: '10'
    DesiredCapacity: '5'
    StartTime: '2015-12-12T00:00:00Z'
```

Example Scheduled-scale-down-specific-time.config

この設定ファイルは、2015-12-12T07:00:00Z にスケールインするよう Elastic Beanstalk に指示しています。

```
option_settings:
  ScheduledScaleDownSpecificTime.aws:autoscaling:scheduledaction:
    MinSize: '1'
    MaxSize: '1'
    DesiredCapacity: '1'
    StartTime: '2015-12-12T07:00:00Z'
```

Example Scheduled-periodic-scale-up.config

この設定ファイルは、毎日午前 9 時にスケールアウトするよう Elastic Beanstalk に指示しています。このアクションは、2015 年 5 月 14 日に開始し、2016 年 1 月 12 日に終了するようにスケジュールされます。

```
option_settings:
  ScheduledPeriodicScaleUp.aws:autoscaling:scheduledaction:
    MinSize: '5'
    MaxSize: '10'
    DesiredCapacity: '5'
    StartTime: '2015-05-14T07:00:00Z'
    EndTime: '2016-01-12T07:00:00Z'
    Recurrence: 0 9 * * *
```

Example Scheduled-periodic-scale-down.config

この設定ファイルでは、毎日午後 6 時に、いずれの実行中のインスタンスにもスケールインしないよう Elastic Beanstalk に指示します。アプリケーションが営業時間外にほぼアイドル状態であることを知っている場合は、同様のスケジュール済みアクションを作成できます。アプリケーションを営業時間外にダウンさせる場合は、`MaxSize` を `0` に変更します。

```
option_settings:
  ScheduledPeriodicScaleDown.aws:autoscaling:scheduledaction:
    MinSize: '0'
    MaxSize: '1'
    DesiredCapacity: '0'
    StartTime: '2015-05-14T07:00:00Z'
    EndTime: '2016-01-12T07:00:00Z'
    Recurrence: 0 18 * * *
```

Example Scheduled-weekend-scale-down.config

この設定ファイルは、毎週金曜日の午後 6 時にスケールインするよう Elastic Beanstalk に指示しています。週末はアプリケーションが大量のトラフィックを受信しないことがわかっている場合は、同様のスケジュールに基づくアクションを作成できます。

```
option_settings:
  ScheduledWeekendScaleDown.aws:autoscaling:scheduledaction:
    MinSize: '1'
    MaxSize: '4'
    DesiredCapacity: '1'
    StartTime: '2015-12-12T07:00:00Z'
    EndTime: '2016-01-12T07:00:00Z'
    Recurrence: 0 18 * * 5
```

Elastic Beanstalk 環境の Auto Scaling ヘルスチェック設定

Amazon EC2 Auto Scaling は、起動する各 Amazon Elastic Compute Cloud (Amazon EC2) インスタンスの状態をモニタリングします。インスタンスが予期せずに終了した場合、Auto Scaling は終了を検出し、代わりのインスタンスを起動します。デフォルトでは、環境用に作成された Auto Scaling グループは Amazon EC2 ステータスチェックを使用します。環境のインスタンスが Amazon EC2 ステータスチェックに失敗した場合、そのインスタンスは停止され、Auto Scaling により置き換えられます。

Amazon EC2 ステータスチェックは、インスタンスで実行されているアプリケーション、サーバー、Docker コンテナの状態ではなく、インスタンスの状態のみを確認するものです。アプリケーションがクラッシュしたが、それが実行されていたインスタンスが正常なままである場合、ロードバランサーの対象から外れる可能性はありますが、Auto Scaling によって自動的に置き換えられることはありません。デフォルトの動作はトラブルシューティングに適しています。アプリケーションがクラッシュしてすぐに Auto Scaling によってインスタンスが置き換えられた場合、起動直後にアプリケーションがクラッシュした場合でも、何が問題だったのかを判別することはできません。

応答を停止したアプリケーションに対応するインスタンスを Auto Scaling で置き換える場合は、設定ファイルを使用して、Elastic Load Balancing ヘルスチェックを使用するように Auto Scaling グループを設定できます。次の例では、Amazon EC2 のステータスチェックに加えて、ロードバランサーのヘルスチェックを使用してインスタンスの状態を確認するようにグループを設定します。

Example .ebextensions/autoscaling.config

```
Resources:
  AWSEBAutoScalingGroup:
    Type: "AWS::AutoScaling::AutoScalingGroup"
    Properties:
      HealthCheckType: ELB
      HealthCheckGracePeriod: 300
```

`HealthCheckType` および `HealthCheckGracePeriod` プロパティの詳細については、AWS CloudFormation ユーザーガイドの「AWS::AutoScaling::AutoScalingGroup」および Amazon EC2 Auto Scaling ユーザーガイドの「Auto Scaling インスタンスのヘルスチェック」を参照してください。

デフォルトでは、Elastic Load Balancing ヘルスチェックは、ポート 80 を経由してインスタンスに TCP 接続を試みるように設定されています。これにより、インスタンス上で実行されているウェブサーバーが接続を受け付けていることが確認されます。ただし、ロードバランサーのヘルスチェックをカスタマイズして、ウェブサーバーだけでなく、アプリケーションが適切な状態になっていることを確認することもできます。猶予期間の設定は、インスタンスがヘルスチェックで失敗した後に、インスタンスを終了して置き換えることなく保留する秒数を設定します。ロードバランサーの対象から外れた後にインスタンスが復旧する可能性があるため、アプリケーションに適した時間をインスタンスに指定します。

Elastic Beanstalk 環境のロードバランサー

ロードバランサーは、環境のインスタンス間でトラフィックを分散します。ロードバランシングを有効にすると、 は環境専用の Elastic Load Balancing ロードバランサー AWS Elastic Beanstalk を作成します。Elastic Beanstalk は、このロードバランサーを完全に管理し、セキュリティ設定を処理し、環境の終了時にロードバランサーを終了します。

また、複数の Elastic Beanstalk 環境間でロードバランサーを共有することもできます。共有ロードバランサーを使用すると、環境ごとに専用のロードバランサーを使用する必要がなくなり、運用コストを節約できます。また、環境で使用する共有ロードバランサーの管理責任は多くなります。

Elastic Load Balancing には、以下のタイプのロードバランサーがあります。

- Classic Load Balancer - 前世代のロードバランサー。HTTP、HTTPS あるいは TCP リクエストトラフィックを環境インスタンスの別のポートにルートします。
- Application Load Balancer - アプリケーション層ロードバランサー。HTTP、HTTPS あるいは HTTPS リクエストトラフィックをリクエストパスに基づいて環境インスタンスの別のポートにルートします。
- Network Load Balancer - ネットワーク層ロードバランサー。TCP リクエストトラフィックを環境インスタンスの別のポートにルートします。アクティブおよびパッシブ両方のヘルスチェックをサポートしています。

Elastic Beanstalk は、3 つのロードバランサータイプすべてをサポートしています。次の表に、2 つの使用パターンのうち、どちらが使用できるタイプかを示します。

ロードバランサーのタイプ	専有	共有済み
Classic Load Balancer	✓ はい	×いいえ
Application Load Balancer	✓ はい	✓ はい
Network Load Balancer	✓ はい	×いいえ

> **Note**
>
> [環境作成] コンソールウィザードでは、Classic Load Balancer (CLB) オプションは無効になっています。既存の環境を Classic Load Balancer で設定している場合は、Elastic

Beanstalk コンソールまたは EB CLI のいずれかを使用して既存の環境を複製することで新しい環境を作成できます。また、EB CLI または を使用して Classic Load Balancer で設定された新しい環境AWS CLIを作成することもできます。これらのコマンドラインツールは、アカウントに CLB がまだ存在しない場合でも、CLB を使用して新しい環境を作成します。

デフォルトでは、Elastic Beanstalk コンソールまたは EB CLI で負荷分散を有効にすると、Elastic Beanstalk によって環境に Application Load Balancer が作成されます。これによって、ロードバランサーはポート 80 で HTTP トラフィックをリッスンし、このトラフィックを同じポートのインスタンスに転送します。環境の作成時にのみに環境が使用するロードバランサーのタイプを選択できます。あとで、実行中の環境のロードバランサーの動作管理の設定は変更できますが、タイプを変更することはできません。

> **Note**
>
> Application Load Balancer を作成するには、2 つ以上のアベイラビリティーゾーンにサブネットを含む VPC にお客様の環境があることが必要です。すべての新しい AWS アカウントには、この要件を満たすデフォルトの VPCs が含まれています。

Elastic Beanstalk がサポートする各ロードバランサーのタイプ、機能、Elastic Beanstalk 環境での設定と管理の方法、および Amazon S3 にアクセスログをアップロードするようにロードバランサーを設定する方法については、以下のトピックを参照してください。

トピック

- Classic Load Balancer の設定
- Application Load Balancer の設定
- 共有 Application Load Balancer の設定
- Network Load Balancer の設定
- アクセスログの設定

Classic Load Balancer の設定

ロードバランシングを有効にすると、 AWS Elastic Beanstalk 環境には Elastic Load Balancing ロードバランサーが備わり、環境内のインスタンス間でトラフィックを分散できます。Elastic Load Balancing は、いくつかのロードバランサータイプをサポートしています。それらについて

は、Elastic Load Balancing ユーザーガイドを参照してください。Elastic Beanstalk では、ロードバランサーを作成したり、作成した共有ロードバランサーを指定したりできます。

このトピックでは、Elastic Beanstalk が作成し、環境専用にする Classic Load Balancer の設定について説明します。Elastic Beanstalk がサポートするすべてのロードバランサータイプの設定については、「Elastic Beanstalk 環境のロードバランサー」を参照してください。

> **Note**
>
> 環境の作成時にのみに環境が使用するロードバランサーのタイプを選択できます。あとで、実行中の環境のロードバランサーの動作管理の設定は変更できますが、タイプを変更することはできません。

序章

Classic Load Balancer は、Elastic Load Balancing の前世代のロードバランサーです。これは、HTTP、HTTPS あるいは TCP リクエストトラフィックを環境インスタンスの別のポートにルートすることをサポートします。

環境で Classic Load Balancer を使用する場合、Elastic Beanstalk によってデフォルトで、ポート 80 の HTTP トラフィックをリッスンし、同じポートのインスタンスに転送するように設定されます。ポート 80 のデフォルトリスナーを削除することはできませんが、無効にすると、トラフィックをブロックすることで同じ機能を実現できます。他のリスナーは追加や削除ができることに注意してください。信頼性に優れた接続をサポートするために、ロードバランサーでポート 443 にリスナーと TLS 証明書を設定できます。

ロードバランサーはヘルスチェックを実行して、アプリケーションを実行している Amazon EC2 インスタンスが正常であるかどうか診断します。ヘルスチェックは、設定した頻度で指定した URL にリクエストします。URL がエラーメッセージを返した場合、または指定したタイムアウト期間内に応答がなかった場合、ヘルスチェックは失敗します。

1 つのサーバーで同じクライアントから複数のリクエストを出させることで、アプリケーションのパフォーマンスが向上する場合は、スティッキーセッションを使用するようロードバランサーを設定できます。スティッキーセッションでは、ロードバランサーはリクエストを出す Amazon EC2 インスタンスを特定する HTTP レスポンスに cookie を追加します。後続のリクエストが同一のクライアントからのものである場合、ロードバランサーは cookie 使用して同じインスタンスにリクエストを送信します。

クロスゾーン負荷分散を使用すると、Classic Load Balancer の各ロードバランサーノードは、有効なすべてのアベイラビリティーゾーンの登録されたインスタンスにリクエストを均等に分散します。クロスゾーン負荷分散が無効の場合は、各ロードバランサーノードは、そのアベイラビリティーゾーンの登録されたインスタンスにのみリクエストを均等に分散します。

インスタンスが正常でなくなったか環境がスケールダウンされたため、インスタンスがロードバランサーから削除された場合は、Connection Draining によってインスタンスとロードバランサー間の接続を閉じる前に、リクエストを完了する時間がインスタンスに与えられます。応答を送信するためにインスタンスに与える時間の長さを変更したり、Connection Drainingを完全に無効化したりできます。

> **Note**
>
> Connection Draining は、Elastic Beanstalk コンソールまたは EB CLI で環境を作成した場合は、デフォルトで有効です。その他のクライアントでは、設定オプションを使って有効にできます。

ロードバランサーの高度な設定を使用して、任意のポートにリスナーを設定し、追加のスティッキーセッションの設定を変更して、EC2 インスタンスに安全に接続するようロードバランサーを設定できます。これらの設定は、設定オプションを通じて使用できます。これは、ソースコードの設定ファイルを使用するか、Elastic Beanstalk API を使用して環境に直接設定できます。これらの設定の多くは、Elastic Beanstalk コンソールでも使用できます。また、Amazon S3 にアクセスログをアップロードするようロードバランサーを設定することもできます。

Elastic Beanstalk コンソールを使用した Classic Load Balancer の設定

Elastic Beanstalk コンソールを使用して、環境の作成時あるいは後の環境の実行時に Classic Load Balancer ポート、HTTPS 証明書、その他の設定を定義できます。

> **Note**
>
> [環境作成] コンソールウィザードでは、Classic Load Balancer (CLB) オプションは無効になっています。既存の環境を Classic Load Balancer で設定している場合は、Elastic Beanstalk コンソールまたは EB CLI のいずれかを使用して既存の環境を複製することで新しい環境を作成できます。また、EB CLI または を使用してAWS CLI、Classic Load Balancer で設定された新しい環境を作成することもできます。これらのコマンドラインツールは、アカウントに CLB がまだ存在しない場合でも、CLB を使用して新しい環境を作成します。

実行中の環境の Classic Load Balancer を Elastic Beanstalk コンソールで設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [ロードバランサー] 設定カテゴリで、[編集] を選択します。

> **Note**
>
> [ロードバランサー] 設定カテゴリに [編集] ボタンがない場合、お客様の環境にはロードバランサーがありません。設定方法については、「環境タイプの変更」を参照してください。

5. Classic Load Balancer の設定に、環境に必要な変更を加えます。
6. ページの最下部で [適用] を選択し変更を保存します。

Classic Load Balancer の設定

- リスナー
- セッション
- クロスゾーンロードバランサー
- Connection Draining
- ヘルスチェック

リスナー

このリストを使用して、ロードバランサーにリスナーを指定します。各リスナーは、指定されたプロトコルを使用して、指定されたポートの着信クライアントトラフィックをインスタンスにルーティングします。初期状態では、このリストにはポート 80 の着信 HTTP トラフィックをポート 80 の HTTP トラフィックをリッスンする環境のインスタンスにルーティングするデフォルトのリスナーが表示されます。

> **Note**
>
> ポート 80 のデフォルトリスナーを削除することはできませんが、無効にすると、トラフィックをブロックすることで同じ機能を実現できます。

既存のリスナーを設定するには

1. テーブルエントリの横にあるチェックボックスをオンにし、[アクション] を選択して、任意のアクションを選択します。
2. [編集] を選択した場合、[Classic Load Balancer listener (クラシックロードバランサーリスナー)] ダイアログボックスを使用して設定を編集し、[保存] を選択します。

たとえば、ロードバランサーがリクエストをそのまま転送するようにするには、デフォルトリスナーを編集して、[プロトコル] を [HTTP] から [TCP] に変更できます。これにより、ロードバランサがヘッダー (`X-Forwarded-For` を含む)を書き換えないようにします。このテクニックは、スティッキーセッションでは機能しません。

リスナーを追加するには

1. [リスナーの追加] を選択します。
2. [Classic Load Balancer listener (クラシックロードバランサーリスナー)] ダイアログボックスで設定を行い、[追加] を選択します。

安全なリスナーの追加は、一般的ユースケースです。次のイメージの例では、ポート 443 の HTTPS トラフィックにリスナーを追加します。このリスナーは、着信したトラフィックをポート 443 で HTTPS トラフィックをリッスンする環境インスタンスサーバーにルーティングします。

HTTPS リスナーを設定する前に、有効な SSL 証明書を保持していることを確認します。次のいずれかを行います：

- リージョンで AWS Certificate Manager (ACM) が利用可能な場合は、ACM を使用して証明書を作成またはインポートします。 AWSACM 証明書のリクエストの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のリクエスト」を参照してください。ACM へのサードパーティー証明書のインポートの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のインポート」を参照してください。
- ACM がお客様の AWS リージョンで利用できない場合は、既存の証明書とキーを IAM にアップロードします。証明書の作成と IAM へのアップロードの詳細については、「IAM ユーザーガイド」の「サーバー証明書の使用」を参照してください。

Elastic Beanstalk における HTTPS の設定と証明書の使用の詳細については、「Elastic Beanstalk 環境の HTTPS の設定」を参照してください。

[SSL 証明書] で、SSL 証明書の ARN を選択します。たとえば、`arn:aws:iam::123456789012:server-certificate/abc/certs/build`、`arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678` などです。

HTTPS の設定と Elastic Beanstalk での証明書の使用の詳細については、「Elastic Beanstalk 環境の HTTPS の設定」を参照してください。

セッション

[セッションの維持が有効です] ボックスをオンまたはオフにして、スティッキーセッションを有効または無効にします。[Cookie の維持期間] を使用して、スティッキーセッションの有効期間を最大 **1000000** 秒までに設定します。[Load balancer ports (ロードバランサーポート)] リストで、デフォルトポリシー (`AWSEB-ELB-StickinessPolicy`) が適用されるリスナーポートを選択します。

クロスゾーンロードバランサー

[複数のアベイラビリティーゾーンでの負荷分散が有効です] ボックスをオンまたはオフにして、クロスゾーン負荷分散を有効あるいは無効にします。

Connection Draining

[Connection Draining が有効です] ボックスをオンまたはオフにして、Connection Draining を有効または無効にします。[Connection Draining] を最大 **3600** 秒に設定します。

ヘルスチェック

次の設定を使用して、ロードバランサーのヘルスチェックを設定します。

- [Health check path (ヘルスチェックパス)] - ロードバランサーがヘルスチェックリクエストを送信するパスです。パスを設定していない場合、ロードバランサーは、状態を確認するためにポート 80 での TCP 接続をするよう試みます。
- [Timeout (タイムアウト)] - ヘルスチェックのレスポンスを待つ時間 (秒単位)。

- [Interval (間隔)] - 個々のインスタンスのヘルスチェックの間隔 (秒単位)。間隔はタイムアウトより大きくする必要があります。
- [Unhealthy threshold (非正常のしきい値)]、[Healthy threshold (正常のしきい値)] - Elastic Load Balancing がインスタンスのヘルス状態を変更する前に、ヘルスチェックに失敗または合格しなければならない回数。

> **Note**
>
> Elastic Load Balancing ヘルスチェックは、環境の Auto Scaling グループのヘルスチェック動作に影響しません。Elastic Load Balancing ヘルスチェックに失敗したインスタンスは、Amazon EC2 Auto Scaling を手動で設定していなければ、Amazon EC2 Auto Scaling によって自動的に置き換えられません。詳細については、「Elastic Beanstalk 環境の Auto Scaling ヘルスチェック設定」を参照してください。

ヘルスチェックと、それが環境の全体的な状態に与える影響の詳細については、「ベーシックヘルスレポート」を参照してください。

EB CLI を使用した Classic Load Balancer の設定

eb create の実行時に、EB CLI によりロードバランサータイプの選択が求められます。

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
1) classic
2) application
3) network
(default is 1):
```

[Enter] を押して、`classic` を選択します。

`--elb-type` オプションを使用してロードバランサータイプを指定することもできます。

```
$ eb create test-env --elb-type classic
```

Classic Load Balancer 設定の名前空間

Classic Load Balancer に関連する設定は、以下の名前空間にあります。

- `aws:elb:healthcheck` - ロードバランサーのヘルスチェックのしきい値、チェックの間隔、およびタイムアウトを設定します。
- `aws:elasticbeanstalk:application` - ヘルスチェック URL の設定
- `aws:elb:loadbalancer` - クロスゾーン負荷分散の有効化 ロードバランサーにセキュリティグループを割り当て、Elastic Beanstalk が作成したデフォルトのセキュリティグループを上書きします。この名前空間には、`aws:elb:listener` 名前空間のオプションによって置き換えられた標準リスナーおよびセキュアリスナーを設定するための廃止されたオプションが含まれます。
- `aws:elb:listener` - ポート 80 にデフォルトのリスナー、ポート 443 にセキュアリスナー、または任意のポートに任意のプロトコルの追加リスナーを設定します。名前空間として `aws:elb:listener` を指定している場合、設定はポート 80 のデフォルトリスナーに適用されます。ポートを指定している場合 (たとえば、`aws:elb:listener:443`) 、リスナーはそのポートに設定されます。
- `aws:elb:policies` - ロードバランサーの追加設定。この名前空間のオプションを使用して、任意のポートにリスナーを設定し、追加のスティッキーセッションの設定を変更して、Amazon EC2 インスタンスに安全に接続するようロードバランサーを設定します。

EB CLI および Elastic Beanstalk コンソールでは、上記のオプションに推奨値が適用されます。設定ファイルを使用して同じファイルを設定する場合は、これらの設定を削除する必要があります。詳細については、「推奨値」を参照してください。

Example .ebextensions/loadbalancer-terminatehttps.config

以下の設定ファイルの例では、ポート 443 の HTTPS リスナーを作成し、ロードバランサーが安全な接続を終了するのに使用する証明書を割り当てて、ポート 80 のデフォルトのリスナーを無効にしています。ロードバランサーは、復号化されたリクエストを HTTP 80 の環境の EC2 インスタンスに転送します。

```
option_settings:
  aws:elb:listener:443:
    ListenerProtocol: HTTPS
```

```
    SSLCertificateId: arn:aws:acm:us-
east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678
    InstancePort: 80
    InstanceProtocol: HTTP
  aws:elb:listener:
    ListenerEnabled: false
```

Application Load Balancer の設定

ロードバランシングを有効にすると、 AWS Elastic Beanstalk 環境には Elastic Load Balancing ロードバランサーが備わり、環境内のインスタンス間でトラフィックを分散できます。Elastic Load Balancing は、いくつかのロードバランサータイプをサポートしています。それらについては、Elastic Load Balancing ユーザーガイドを参照してください。Elastic Beanstalk では、ロードバランサーを作成したり、作成した共有ロードバランサーを指定したりできます。

このトピックでは、Elastic Beanstalk が作成し、環境専用にする Application Load Balancer の設定について説明します。「the section called “共有 Application Load Balancer”」も参照してください。Elastic Beanstalk がサポートするすべてのロードバランサータイプの設定については、「the section called “ロードバランサー”」を参照してください。

> **Note**
>
> 環境の作成時にのみに環境が使用するロードバランサーのタイプを選択できます。実行中の環境のロードバランサーの行動管理の設定は変更できますが、タイプを変更することはできません。また、専用ロードバランサーから共有ロードバランサーに切り替えることも、その逆もできません。

序章

Application Load Balancer は、アプリケーションネットワークプロトコルレイヤーでトラフィックを検査してリクエストのパスを識別し、パスが異なるリクエストを別々の送信先にダイレクトできるようにします。

環境で Application Load Balancer を使用する場合、Elastic Beanstalk によってデフォルトで、Classic Load Balancer と同じ機能を実行するように設定されます。デフォルトのリスナーはポート 80 で HTTP リクエストを受け取り、環境内のインスタンスに分散します。ポート 443 でセキュアなリスナーを証明書と共に追加して、HTTPS トラフィックの復号、ヘルスチェックの動作の

設定、ロードバランサーから Amazon Simple Storage Service (Amazon S3) バケットへのアクセスログのプッシュを行うことができます。

> **Note**
>
> Classic Load Balancer や Network Load Balancer とは異なり、Application Load Balancer では、トランスポート層 (レイヤー 4) の TCP または SSL/TLS リスナーを配置できません。これは HTTP および HTTPS リスナーでのみサポートされています。また、これはロードバランサーとバックエンドインスタンス間の HTTPS 接続を認証するバックエンド認証を使用できません。

Elastic Beanstalk 環境では、Application Load Balancer を使用して、特定のパスのトラフィックをウェブサーバーインスタンス上の別のプロセスに転送できます。Classic Load Balancer では、リスナーへのすべてのトラフィックはバックエンドインスタンスの 1 つのプロセスにルーティングされます。Application Load Balancer を使用すると、リスナーで複数のルールを設定し、特定のプロセスへのリクエストを別々のバックエンドポートにルーティングできます。各プロセスは、プロセスがリッスンするポートで設定します。

たとえば、ログインプロセスをメインアプリケーションとは別に実行できます。環境のインスタンスのメインアプリケーションがほとんどのリクエストを受け入れ、ポート 80 でリッスンしている間、ログインプロセスはポート 5000 でリッスンし、`/login` パスへのリクエストを受け入れます。クライアントからのすべての着信リクエストは、ポート 80 に入ります。Application Load Balancer を使用すると、ポート 80 の着信トラフィック用に、リクエスト内のパスに応じて 2 つの異なるプロセスにトラフィックをルーティングする 2 つのルールで単一リスナーを設定できます。ポート 5000 でリッスンするログインプロセスに、`/login` へのトラフィックをルーティングするカスタムルールを追加します。デフォルトのルールは、他のすべてのトラフィックをポート 80 でリッスンするメインアプリケーションプロセスにルートします。

Application Load Balancer ルールにより、リクエストがターゲットグループにマッピングされます。Elastic Beanstalk では、ターゲットグループはプロセスで表されます。プロセスのプロトコル、ポート、ヘルスチェック設定を構成できます。プロセスは環境内のインスタンスで実行されるプロセスを表します。デフォルトプロセスは、アプリケーションの前面で実行されるリバースプロキシ (nginx または Apache) のポート 80 のリスナーです。

> **Note**
>
> Elastic Beanstalk の外部では、ターゲットグループはインスタンスのグループにマッピングされます。リスナーはルールとターゲットグループを使用して、パスに基づいてトラフィッ

クを別のインスタンスにルーティングできます。Elastic Beanstalk 内で、環境のすべてのインスタンスは同じであるため、別々のポートでリッスンしているプロセス間で区別されます。

Classic Load Balancer では、環境全体に対して 1 つのヘルスチェックパスが使用されます。Application Load Balancer では、プロセス別に、ロードバランサーと Elastic Beanstalk の拡張ヘルスモニタリングによってモニタリングされるヘルスチェックパスがあります。

Application Load Balancer を使用するには、デフォルトまたはカスタム VPC に環境があり、サービスロールにアクセス許可の標準セットがあることが必要です。古いサービスロールがある場合は、そこでのアクセス許可を更新して、`elasticloadbalancing:DescribeTargetHealth` および `elasticloadbalancing:DescribeLoadBalancers` を含めなければならない可能性があります。Application Load Balancer の詳細については、「Application Load Balancer とは」を参照してください。

> **Note**
>
> Application Load Balancer のヘルスチェックでは、Elastic Beanstalk のヘルスチェックパスは使用されません。代わりに、各プロセス用に別々に設定された特定のパスを使用します。

Elastic Beanstalk コンソールを使用した Application Load Balancer の設定

Elastic Beanstalk コンソールを使用して、環境の作成中か、後で環境の実行中に、Application Load Balancer のリスナー、プロセス、ルールを設定できます。

環境の作成中に Elastic Beanstalk コンソールで Application Load Balancer を設定するには

1. Elastic Beanstalk 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 [Environments (環境)] を選択します。
3. [新しい環境の作成] を選択して、環境の作成を開始します。
4. ウィザードのメインページで、[環境の作成] を選択する前に、[さらにオプションを設定] を選択します。
5. [高可用性] 設定プリセットを選択します。

   または、[容量] 設定カテゴリで [ロードバランサー] 環境タイプを設定します。詳細については、「容量」を参照してください。

6. [ロードバランサー] 設定カテゴリで、[編集] を選択します。
7. [Application Load Balancer] および [Dedicated (専用)] オプションがまだ選択されていない場合は、それらのオプションを選択します。
8. Application Load Balancer の設定に、環境に必要な変更を加えます。
9. [保存] を選択し、環境に必要なその他の任意の設定変更を行います。
10. [環境を作成] を選択します。

実行中の環境の Application Load Balancer を Elastic Beanstalk コンソールで設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [ロードバランサー] 設定カテゴリで、[編集] を選択します。

> **Note**
>
> [ロードバランサー] 設定カテゴリに [編集] ボタンがない場合、お客様の環境にはロードバランサーがありません。設定方法については、「環境タイプの変更」を参照してください。

5. Application Load Balancer の設定に、環境に必要な変更を加えます。
6. ページの最下部で [適用] を選択し変更を保存します。

Application Load Balancer の設定

- リスナー
- プロセス
- ルール
- アクセスログのキャプチャ

リスナー

このリストを使用して、ロードバランサーにリスナーを指定します。各リスナーは、指定されたプロトコルを使用して、指定されたポートの着信クライアントトラフィックをインスタンスの 1 つ以上

のプロセスにルーティングします。初期状態では、このリストには、ポート 80 で着信する HTTP トラフィックを、[デフォルト] という名前のプロセスにルーティングするデフォルトのリスナーが表示されます。

既存のリスナーを設定するには

1. そのテーブルエントリの横にあるチェックボックスを選択し、続いて [アクション]、[編集] の順に選択します。
2. [Application Load Balancer listener (アプリケーションロードバランサーリスナー)] ダイアログボックスを使用して設定を編集し、[保存] を選択します。

リスナーを追加するには

1. [リスナーの追加] を選択します。
2. [Application Load Balancer リスナー] ダイアログボックスで必要な設定を行い、[追加] を選択します。

[Application Load Balancer リスナー] ダイアログボックスを使用して、リスナーがトラフィックをリッスンするポートおよびプロトコル、およびトラフィックのルーティング先となるプロセスを選択します。HTTPS プロトコルを選択する場合、SSL 設定を構成します。

HTTPS リスナーを設定する前に、有効な SSL 証明書を保持していることを確認します。次のいずれかを行います：

- リージョンで AWS Certificate Manager (ACM) が使用可能な場合は、ACM を使用して証明書を作成またはインポートします。 AWSACM 証明書のリクエストの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のリクエスト」を参照してください。ACM へのサードパーティー証明書のインポートの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のインポート」を参照してください。
- ACM がお客様の AWS リージョンで利用できない場合は、既存の証明書とキーを IAM にアップロードします。証明書の作成と IAM へのアップロードの詳細については、「IAM ユーザーガイド」の「サーバー証明書の使用」を参照してください。

Elastic Beanstalk における HTTPS の設定と証明書の使用の詳細については、「Elastic Beanstalk 環境の HTTPS の設定」を参照してください。

プロセス

このリストを使用して、ロードバランサーにプロセスを指定します。プロセスは、トラフィックをルートするターゲットです。各リスナーは、指定されたプロトコルを使用して、指定されたポートの着信クライアントトラフィックをインスタンスの 1 つ以上のプロセスにルーティングします。初期状態では、このリストにはポート 80 で着信する HTTP トラフィックをリッスンするデフォルトのプロセスが表示されます。

既存のプロセスの設定を編集するか、あるいは新しいプロセスを追加します。リストのプロセスの編集またはプロセスの追加を開始するには、リスナーリストと同じステップを使用します。[環境プロセス] ダイアログボックスが開きます。

Application Load Balancer の環境プロセスダイアログボックスでの設定

- 定義
- ヘルスチェック
- セッション

定義

これらの設定を使用して、[名前] と、リクエストをリッスンする [ポート] および [プロトコル] でプロセスを定義します。

ヘルスチェック

次の設定を使用して、プロセスのヘルスチェックを設定します。

- [HTTP code (HTTP コード)] - 正常なプロセスを示す HTTP ステータスコード。
- [Path (パス)] - プロセスのヘルスチェックリクエストパス。
- [Timeout (タイムアウト)] - ヘルスチェックのレスポンスを待つ時間 (秒単位)。
- [Interval (間隔)] - 個々のインスタンスのヘルスチェックの間隔 (秒単位)。間隔はタイムアウトより大きくする必要があります。
- [Unhealthy threshold (非正常のしきい値)]、[Healthy threshold (正常のしきい値)] - Elastic Load Balancing がインスタンスのヘルス状態を変更する前に、ヘルスチェックに失敗または合格しなければならない回数。
- [Deregistration delay (登録解除の遅延)] - インスタンスの登録を解除する前にアクティブリクエストの完了を待機する時間 (秒単位)。

> **Note**
>
> Elastic Load Balancing ヘルスチェックは、環境の Auto Scaling グループのヘルスチェック動作に影響しません。Elastic Load Balancing ヘルスチェックに失敗したインスタンスは、Amazon EC2 Auto Scaling を手動で設定していなければ、Amazon EC2 Auto Scaling によって自動的に置き換えられません。詳細については、「Elastic Beanstalk 環境の Auto Scaling ヘルスチェック設定」を参照してください。

ヘルスチェックと、それが環境の全体的な状態に与える影響の詳細については、「ベーシックヘルスレポート」を参照してください。

セッション

[Stickiness policy enabled] ボックスをオンまたはオフにして、スティッキーセッションを有効または無効にします。[Cookie の維持期間] を使用して、スティッキーセッションの有効期間を最大 **604800** 秒までに設定します。

ルール

このリストを使用して、ロードバランサーにカスタムリスナールールを指定します。ルールマップでは、リスナーがターゲットプロセスへの特定のパスパターンを受信することをリクエストします。各リスナーでは、インスタンスでの異なるプロセスへの別々のパスにリクエストをルーティングする複数のルールを保持することができます。

ルールには着信するリクエストに適用されるべき優先度を決定する優先順位の数字があります。追加する新しいリスナーごとに、Elastic Beanstalk はリスナーのトラフィックのすべてをデフォルトのプロセスにルーティングするデフォルトのルールを追加します。このデフォルトのルールの優先順位は最も低く、着信するリクエストに一致する他のルールが同じリスナーにない場合にのみ適用されます。最初は、カスタムルールを追加していない場合、リストは空です。すべてのリスナーのデフォルトルールは表示されません。

既存のルールの設定を編集するか、あるいは新しいルールを追加します。リストのルールの編集またはルールの追加を開始するには、リスナーリストと同じ手順を使用します。[リスナールール] ダイアログボックスが開き、以下の設定が表示されます。

- [Name (名前)] - ルールの名前。
- [Listener port (リスナーポート)] - ルールが適用されるリスナーのポート。
- [Priority (優先度)] - ルールの優先度。数の小さい優先度番号が優先されます。リスナーのルールの優先順位は一意である必要があります。
- Match conditions (一致条件) - ルールが適用されるリクエスト URL の条件のリスト。条件には、[HosTheader] (URL のドメイン部分) と [PathPattern] (URL のパス部分) の 2 種類があります。条件は最大 5 つまで追加できます。各条件値は最大 128 文字で、ワイルドカード文字を含めることができます。
- [Process (プロセス)] - ロードバランサーがルールと一致するリクエストをルーティングするプロセス。

既存のルールを編集する場合、その [名前] および [リスナーポート] を変更することはできません。

アクセスログのキャプチャ

これらの設定を使用して、Application Load Balancer に送信されたリクエストの詳細情報のログを取得するように Elastic Load Balancing を設定します。アクセスログのキャプチャはデフォルトでは無効になっています。[Store logs (ログの保存)] が有効なとき、Elastic Load Balancing は、設定された S3 バケットにログを保存します。[Prefix (プレフィックス)] 設定は、ログのバケットの最上位フォルダを指定します。Elastic Load Balancing は、プレフィックスの下の `AWSLogs` という名前のフォルダにログを配置します。プレフィックスを指定しない場合、Elastic Load Balancing はバケットのルートレベルにそのフォルダを配置します。

> **Note**
>
> アクセスログキャプチャ用に設定した Amazon S3 バケットが、Elastic Beanstalk がアカウント用に作成したバケットでない場合は、 AWS Identity and Access Management (IAM) ユーザーに適切なアクセス許可を持つユーザーポリシーを追加してください。Elastic Beanstalk が提供する管理ユーザーポリシーは、Elastic Beanstalk で管理されるリソースに対するアクセス許可のみを対象としています。

アクセス許可やその他の要件を含むアクセスログの詳細については、「Access logs for your Application Load Balancer (アプリケーションロードバランサーのアクセスログ) 」を参照してください。

例: セキュアなリスナーと 2 つのプロセスを使用する Application Load Balancer

この例では、アプリケーションにはエンドツーエンドのトラフィック暗号および管理リクエストを処理するための別のプロセスが必要です。

これらの要件を満たすように環境の Application Load Balancer を設定するには、デフォルトのリスナーを削除し、HTTPS リスナーを追加して、デフォルトのプロセスが HTTPS のポート 443 をリッスンするように指定します。また、別のパスに管理トラフィックのプロセスおよびリスナールールを追加します。

この例でロードバランサーを設定するには

1. 安全なリスナーを追加します。 [ポート] に「**443**」と入力します。[プロトコル] で、[**HTTPS**] を選択します。[SSL 証明書] で、SSL 証明書の ARN を選択します。たとえば、**arn:aws:iam::123456789012:server-certificate/abc/certs/build**、**arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678** などです。

   [デフォルトプロセス] で、**default** を選択したままにします。

   これでリストに追加のリスナーが表示されます。

2. デフォルトのポート 80 HTTP リスナーを無効にします。デフォルトのリスナーについて、[有効] オプションをオフにします。

3. デフォルトプロセスを HTTPS に設定します。デフォルトのプロセスを選択し、[アクション] で [編集] を選択します。 [ポート] に「**443**」と入力します。[プロトコル] で、[**HTTPS**] を選択します。

4. 管理プロセスを追加します。[Name] (名前) に、「**admin**」と入力します。 [ポート] に「**443**」と入力します。[プロトコル] で、[**HTTPS**] を選択します。[ヘルスチェック] で [パス] に「**/admin**」と入力します。

5. 管理トラフィックのルールを追加します。[Name] (名前) に、「**admin**」と入力します。[リスナーポート] で **443** と入力します。[一致条件] には、**/admin/*** の値で [PathPattern] を追加します。[プロセス] で、**admin** を選択します。

EB CLI を使用した Application Load Balancer の設定

eb create の実行時に、EB CLI によりロードバランサータイプの選択が求められます。

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
1) classic
2) application
3) network
(default is 2):
```

`--elb-type` オプションでロードバランサータイプを指定することもできます。

```
$ eb create test-env --elb-type application
```

Application Load Balancer の名前空間

Application Load Balancer に関連する設定は、以下の名前空間にあります。

- `aws:elasticbeanstalk:environment` - 環境のロードバランサーのタイプを選択します。Application Load Balancer の値は `application` です。

  設定ファイル (.Ebextensions) では、このオプションを設定できません。
- `aws:elbv2:loadbalancer` - Application Load Balancer 全体に適用されるアクセスログとその他の設定を定義します。
- `aws:elbv2:listener` - Application Load Balancer のリスナーを設定します。これらの設定は、Classic Load Balancer に対する `aws:elb:listener` の設定にマッピングされます。
- `aws:elbv2:listenerrule` - リクエストパスに応じて、別々のプロセスにトラフィックをルーティングするルールを設定します。ルールは Application Load Balancer に固有です。
- `aws:elasticbeanstalk:environment:process` - ヘルスチェックを設定し、環境のインスタンスで実行するプロセス用のポートとプロトコルを指定します。ポートとプロト

コル設定は Classic Load Balancer のリスナー用の `aws:elb:listener` のインスタンスポートおよびインスタンスプロトコル設定にマッピングされます。ヘルスチェックの設定は、`aws:elb:healthcheck` および `aws:elasticbeanstalk:application` 名前空間の設定にマッピングされます。

Example .ebextensions/alb-access-logs.config

以下の設定ファイルでは、Application Load Balancer を使用した環境用のアクセスログのアップロードが可能になります。

```
option_settings:
  aws:elbv2:loadbalancer:
    AccessLogsS3Bucket: amzn-s3-demo-bucket
    AccessLogsS3Enabled: 'true'
    AccessLogsS3Prefix: beanstalk-alb
```

Example .ebextensions/alb-default-process.config

以下の設定ファイルでは、デフォルトプロセスのヘルスチェックとセッション維持の設定を変更します。

```
option_settings:
  aws:elasticbeanstalk:environment:process:default:
    DeregistrationDelay: '20'
    HealthCheckInterval: '15'
    HealthCheckPath: /
    HealthCheckTimeout: '5'
    HealthyThresholdCount: '3'
    UnhealthyThresholdCount: '5'
    Port: '80'
    Protocol: HTTP
    StickinessEnabled: 'true'
    StickinessLBCookieDuration: '43200'
```

Example .ebextensions/alb-secure-listener.config

以下の設定ファイルでは、ポート 443 でセキュアリスナーと一致するプロセスを追加します。

```
option_settings:
  aws:elbv2:listener:443:
```

```
    DefaultProcess: https
    ListenerEnabled: 'true'
    Protocol: HTTPS
    SSLCertificateArns: arn:aws:acm:us-
east-2:123456789012:certificate/21324896-0fa4-412b-bf6f-f362d6eb6dd7
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
    Protocol: HTTPS
```

Example .ebextensions/alb-admin-rule.config

以下の設定ファイルでは、ポート 4443 でリッスンする /admin というプロセスに、リクエストパス admin を使用してトラフィックをルーティングするルールでセキュアリスナーを追加します。

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
    Protocol: HTTPS
    Rules: admin
    SSLCertificateArns: arn:aws:acm:us-
east-2:123456789012:certificate/21324896-0fa4-412b-bf6f-f362d6eb6dd7
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
    Protocol: HTTPS
  aws:elasticbeanstalk:environment:process:admin:
    HealthCheckPath: /admin
    Port: '4443'
    Protocol: HTTPS
  aws:elbv2:listenerrule:admin:
    PathPatterns: /admin/*
    Priority: 1
    Process: admin
```

共有 Application Load Balancer の設定

ロードバランシングを有効にすると、 AWS Elastic Beanstalk 環境内のインスタンス間でトラフィックを分散するための Elastic Load Balancing ロードバランサーが環境に組み込まれます。Elastic Load Balancing は、いくつかのロードバランサータイプをサポートしています。それらについては、Elastic Load Balancing ユーザーガイドを参照してください。Elastic Beanstalk では、ロードバランサーを作成したり、作成した共有ロードバランサーを指定したりできます。

このトピックでは、共有 Application Load Balancer を作成して環境に関連付ける設定について説明します。「the section called “Application Load Balancer”」も参照してください。Elastic Beanstalk がサポートするすべてのロードバランサータイプの設定については、「Elastic Beanstalk 環境のロードバランサー」を参照してください。

> **Note**
>
> 環境の作成時にのみに環境が使用するロードバランサーのタイプを選択できます。実行中の環境のロードバランサーの行動管理の設定は変更できますが、タイプを変更することはできません。また、専用ロードバランサーから共有ロードバランサーに切り替えることも、その逆もできません。

序章

共有ロードバランサーは、Amazon Elastic Compute Cloud (Amazon EC2) サービスにより自分で作成および管理し、複数の Elastic Beanstalk 環境で使用するロードバランサーです。

負荷分散されたスケーリング環境を作成し、Application Load Balancer の使用を選択すると、Elastic Beanstalk はデフォルトで環境専用のロードバランサーを作成します。Application Load Balancer とは何か、Elastic Beanstalk 環境でのしくみについては、Elastic Beanstalk 用の Application Load Balancer の設定の概要を参照してください。

状況によっては、複数の専用ロードバランサーを使用するコストを削減したい場合があります。これは、アプリケーションがモノリシックサービスではなくマイクロサービスのスイートである場合など、複数の環境がある場合に役立ちます。このような場合は、共有ロードバランサーの使用を選択できます。

共有ロードバランサーを使用するには、まず Amazon EC2 で共有ロードバランサーを作成し、1 つ以上のリスナーを追加します。Elastic Beanstalk 環境の作成中に、ロードバランサーを提供し、リスナーポートを選択します。Elastic Beanstalk は、リスナーを環境内のデフォルトのプロセスに関連付けます。カスタムリスナールールを追加して、特定のホストヘッダーとパスから他の環境プロセスにトラフィックをルーティングできます。

Elastic Beanstalk は、共有ロードバランサーにタグを追加します。タグ名は `elasticbeanstalk:shared-elb-environment-count` で、値はこのロードバランサーを共有する環境の数です。

共有ロードバランサーの使用は、専有ロードバランサーの使用とはいくつかの方法で異なります。

内容	専用 Application Load Balancer	共有 Application Load Balancer
管理	Elastic Beanstalk は、ロードバランサー、リスナー、リスナールール、プロセス (ターゲットグループ) を作成および管理します。Elastic Beanstalk は、環境の終了時にそれらも削除します。そのオプションを選択した場合、Elastic Beanstalk はロードバランサーのアクセスログのキャプチャを設定できます。	Elastic Beanstalk の外部でロードバランサーとリスナーを作成および管理します。Elastic Beanstalk によってデフォルトのルールとデフォルトのプロセスが作成および管理されます。お客様はルールとプロセスを追加できます。Elastic Beanstalk は、環境の作成中に追加されたリスナールールとプロセスを削除します。
リスナールール	Elastic Beanstalk は、リスナーごとにデフォルトのルールを作成し、すべてのトラフィックをリスナーのデフォルトのプロセスにルーティングします。	Elastic Beanstalk は、デフォルトのルールをポート 80 リスナーにのみ関連付けます (存在する場合)。別のデフォルトのリスナーポートを選択した場合は、デフォルトのルールをそれに関連付ける必要があります (Elastic Beanstalk コンソールと EB CLI を使用して行います)。 ロードバランサーを共有する環境間でリスナールール条件の競合を解決するために、Elastic Beanstalk はホストヘッダー条件として環境の CNAME をリスナールールに追加します。 Elastic Beanstalk は、ロードバランサーを共有する環境間でルールの優先順位設定を相対的なものとして扱い、作成時に絶対的な優先順位にマッピングします。
セキュリティグループ	Elastic Beanstalk は、デフォルトのセキュリティグループを作成し、ロードバランサーにアタッチします。	ロードバランサーに使用する 1 つ以上のセキュリティグループを設定できます。設定しない場合、Elastic Beanstalk は、管理している既存のセキュリティグループがロードバランサーに既にアタッチされているかどうかを確認します。アタッチされていない場合、Elastic Beanstalk

内容	専用 Application Load Balancer	共有 Application Load Balancer
		はセキュリティグループを作成し、ロードバランサーにアタッチします。Elastic Beanstalk は、ロードバランサーを共有する最後の環境が終了すると、このセキュリティグループを削除します。
更新	環境の作成後に Application Load Balancer を更新できます。リスナー、リスナールール、およびプロセスを編集できます。ロードバランサーのアクセスログのキャプチャを設定できます。	Elastic Beanstalk を使用して Application Load Balancer でアクセスログのキャプチャを設定することはできません。また、環境の作成後にリスナーとリスナールールを更新することもできません。プロセス (ターゲットグループ) のみを更新できます。アクセスログのキャプチャを設定し、リスナーとリスナールールを更新するには、Amazon EC2 を使用します。

Elastic Beanstalk コンソールを使用した共有 Application Load Balancer の設定

Elastic Beanstalk コンソールを使用して、環境の作成中に共有 Application Load Balancer を設定できます。環境で使用するアカウントの共有可能なロードバランサーの 1 つを選択し、デフォルトのリスナーポートを選択して、追加のプロセスとリスナールールを設定できます。

環境の作成後は、Application Load Balancer コンソールで共有 Application Load Balancer の設定を編集することはできません。リスナー、リスナールール、プロセス (ターゲットグループ)、アクセスログのキャプチャを設定するには、Amazon EC2 を使用します。

環境の作成中に Elastic Beanstalk コンソールで Application Load Balancer を設定するには

1. Elastic Beanstalk 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 [Environments (環境)] を選択します。
3. [新しい環境の作成] を選択して、環境の作成を開始します。
4. ウィザードのメインページで、[環境の作成] を選択する前に、[さらにオプションを設定] を選択します。
5. [高可用性] 設定プリセットを選択します。

または、[容量] 設定カテゴリで [ロードバランサー] 環境タイプを設定します。詳細については、「容量」を参照してください。

6. [ロードバランサー] 設定カテゴリで、[編集] を選択します。
7. まだ選択されていない場合は、[Application Load Balancer] オプションを選択してから、[Shared (共有)] オプションを選択します。
8. 共有 Application Load Balancer 設定に、環境に必要な変更を加えます。
9. [保存] を選択し、環境に必要なその他の任意の設定変更を行います。
10. [環境を作成] を選択します。

共有 Application Load Balancer の設定

- 共有 Application Load Balancer
- プロセス
- ルール

共有 Application Load Balancer

このセクションを使用して、環境に合わせて共有 Application Load Balancer を選択し、デフォルトのトラフィックルーティングを設定します。

ここで共有 Application Load Balancer を設定する前に、Amazon EC2 を使用してアカウントで、1 つ以上のリスナーと共有する 1 つ以上の Application Load Balancer を定義します。まだ選択されていない場合は、[Manage load balancers (ロードバランサーの管理)] を選択できます。Elastic Beanstalk は、Amazon EC2 コンソールを新しいブラウザタブで開きます。

Elastic Beanstalk の外部で共有ロードバランサーを設定した場合、このコンソールセクションで以下のように設定します。

- [Load balancer ARN (ロードバランサー ARN)] - この環境で使用する共有ロードバランサー。ロードバランサーのリストから選択するか、ロードバランサーの Amazon リソースネーム (ARN) を入力します。
- [Default listener port (デフォルトのリスナーポート)] - 共有ロードバランサーがリッスンするリスナーポート。既存のリスナーポートのリストから選択します。ホストヘッダーに環境の CNAME を含む、このリスナーからのトラフィックは、この環境のデフォルトプロセスにルーティングされます。

プロセス

このリストを使用して、共有ロードバランサーにプロセスを指定します。プロセスは、トラフィックをルートするターゲットです。初期状態では、このリストには、デフォルトリスナーからトラフィックを受信するデフォルトプロセスが表示されます。

既存のプロセスを設定するには

1. そのテーブルエントリの横にあるチェックボックスを選択し、続いて [アクション]、[編集] の順に選択します。
2. [環境プロセス] ダイアログボックスを使用して設定を編集し、[保存] を選択します。

プロセスを追加するには

1. [プロセスの追加] を選択します。
2. [環境プロセス] ダイアログボックスで設定を行い、[追加] を選択します。

Application Load Balancer の環境プロセスダイアログボックスでの設定

- 定義
- ヘルスチェック
- セッション

定義

これらの設定を使用して、[名前] と、リクエストをリッスンする [ポート] および [プロトコル] でプロセスを定義します。

ヘルスチェック

次の設定を使用して、プロセスのヘルスチェックを設定します。

- [HTTP code (HTTP コード)] - 正常なプロセスを示す HTTP ステータスコード。
- [Path (パス)] - プロセスのヘルスチェックリクエストパス。
- [Timeout (タイムアウト)] - ヘルスチェックのレスポンスを待つ時間 (秒単位)。

- [Interval (間隔)] - 個々のインスタンスのヘルスチェックの間隔 (秒単位)。間隔はタイムアウトより大きくする必要があります。
- [Unhealthy threshold (非正常のしきい値)]、[Healthy threshold (正常のしきい値)] - Elastic Load Balancing がインスタンスのヘルス状態を変更する前に、ヘルスチェックに失敗または合格しなければならない回数。
- [Deregistration delay (登録解除の遅延)] - インスタンスの登録を解除する前にアクティブリクエストの完了を待機する時間 (秒単位)。

> **Note**
>
> Elastic Load Balancing ヘルスチェックは、環境の Auto Scaling グループのヘルスチェック動作に影響しません。Elastic Load Balancing ヘルスチェックに失敗したインスタンスは、Amazon EC2 Auto Scaling を手動で設定していなければ、Amazon EC2 Auto Scaling によって自動的に置き換えられません。詳細については、「Elastic Beanstalk 環境の Auto Scaling ヘルスチェック設定」を参照してください。

ヘルスチェックと、それが環境の全体的な状態に与える影響の詳細については、「ベーシックヘルスレポート」を参照してください。

セッション

[Stickiness policy enabled] ボックスをオンまたはオフにして、スティッキーセッションを有効または無効にします。[Cookie の維持期間] を使用して、スティッキーセッションの有効期間を最大 **604800** 秒までに設定します。

ルール

このリストを使用して、共有ロードバランサーにカスタムリスナールールを指定します。ルールマップでは、リスナーがターゲットプロセスへの特定のパスパターンを受信することをリクエストします。各リスナーでは、リスナーを共有する異なる環境のインスタンスでの異なるプロセスへの別々のパスにリクエストをルーティングする複数のルールを保持することができます。

ルールには着信するリクエストに適用されるべき優先度を決定する優先順位の数字があります。Elastic Beanstalk は、すべてのデフォルトのリスナーのトラフィックを新しい環境のデフォルトのプロセスにルーティングするデフォルトのルールを追加します。このデフォルトのルールの優先順

位は最も低く、着信するリクエストに一致する他のルールが同じリスナーにない場合にのみ適用されます。最初は、カスタムルールを追加していない場合、リストは空です。デフォルトのルールは表示されません。

既存のルールの設定を編集するか、あるいは新しいルールを追加します。リストのルールの編集またはルールの追加を開始するには、プロセスリストと同じ手順を使用します。[リスナールール] ダイアログボックスが開き、以下の設定が表示されます。

- [Name (名前)] - ルールの名前。
- [Listener port (リスナーポート)] - ルールが適用されるリスナーのポート。
- [Priority (優先度)] - ルールの優先度。数の小さい優先度番号が優先されます。リスナーのルールの優先順位は一意である必要があります。Elastic Beanstalk では、共有環境全体でルールの優先順位が相対的なものとして扱われ、作成時に絶対的な優先順位にマッピングされます。
- Match conditions (一致条件) - ルールが適用されるリクエスト URL の条件のリスト。条件には、[HosTheader] (URL のドメイン部分) と [PathPattern] (URL のパス部分) の 2 種類があります。1 つの条件は環境サブドメイン用に予約されており、最大 4 つの条件を追加できます。各条件値は最大 128 文字で、ワイルドカード文字を含めることができます。
- [Process (プロセス)] - ロードバランサーがルールと一致するリクエストをルーティングするプロセス。

例: セキュアなマイクロサービスベースのアプリケーションに共有 Application Load Balancer を使用する

この例では、アプリケーションは複数のマイクロサービスで構成され、それぞれが Elastic Beanstalk 環境として実装されています。さらに、エンドツーエンドのトラフィック暗号化が必要です。ここでは、マイクロサービス環境の 1 つを紹介します。マイクロサービス環境には、ユーザーリクエスト用のメインプロセスと、管理リクエストを処理するための別個のプロセスがあります。

これらの要件を満たすには、Amazon EC2 を使用して、マイクロサービス間で共有する Application Load Balancer を作成します。ポート 443 および HTTPS プロトコルに安全なリスナーを追加します。次に、マイクロサービスドメインごとに 1 つずつ、複数の SSL 証明書をリスナーに追加します。Application Load Balancer とセキュアなリスナーの作成の詳細については、「Application Load Balancer ユーザーガイド」の「Application Load Balancer の作成」と「Application Load Balancer 用の HTTPS リスナーを作成する」を参照してください。

Elastic Beanstalk で、共有 Application Load Balancer を使用するように各マイクロサービス環境を設定してから、デフォルトのリスナーポートを 443 に設定します。ここで示している特定の環境の場合、デフォルトのプロセスが HTTPS でポート 443 をリッスンしていることを示し、別のパスで管理トラフィックのプロセスとリスナールールを追加します。

この例で共有ロードバランサーを設定するには

1. [Shared Application Load Balancer (共有 Application Load Balancer)] セクションでロードバランサーを選択してから、[Default listener port (デフォルトのリスナーポート)] で **443** を選択します。ロードバランサーが唯一のリスナーである場合は、リスナーポートが既に選択されている必要があります。

2. デフォルトプロセスを HTTPS に設定します。デフォルトのプロセスを選択し、[アクション] で [編集] を選択します。[Port (ポート)] に「**443**」と入力します。[プロトコル] で、[**HTTPS**] を選択します。

3. 管理プロセスを追加します。[名前] に **admin** と入力します。[Port (ポート)] に「**443**」と入力します。[プロトコル] で、[**HTTPS**] を選択します。[ヘルスチェック] で [パス] に **/admin** と入力します。

4. 管理トラフィックのルールを追加します。[名前] に **admin** と入力します。[リスナーポート] に **443** と入力します。[一致条件] には、**/admin/*** の値で [PathPattern] を追加します。[プロセス] で、**admin** を選択します。

EB CLI を使用した共有 Application Load Balancer の設定

eb create の実行時に、EB CLI によりロードバランサータイプの選択が求められます。`application` (デフォルト) を選択し、アカウントに共有可能な Application Load Balancer が 1 つ以上ある場合、EB CLI では共有 Application Load Balancer を使用するかどうかも尋ねられます。**y** と答えると、ロードバランサーとデフォルトポートを選択するように求められます。

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF
```

```
Select a load balancer type
1) classic
2) application
3) network
(default is 2):

Your account has one or more sharable load balancers. Would you like your new
 environment to use a shared load balancer?(y/N) y

Select a shared load balancer
1)MySharedALB1 - arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/app/
MySharedALB1/6d69caa75b15d46e
2)MySharedALB2 - arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/app/
MySharedALB2/e574ea4c37ad2ec8
(default is 1): 2

Select a listener port for your shared load balancer
1) 80
2) 100
3) 443
(default is 1): 3
```

コマンドオプションを使用して、共有ロードバランサーを指定することもできます。

```
$ eb create test-env --elb-type application --shared-lb MySharedALB2 --shared-lb-
port 443
```

共有 Application Load Balancer の名前空間

共有 Application Load Balancer に関連する設定は、以下の名前空間にあります。

- `aws:elasticbeanstalk:environment` - 環境のロードバランサーの種類を選択し、Elastic Beanstalk に共有ロードバランサーを使用することを指示します。

  設定ファイル (.Ebextensions) では、これらの 2 つのオプションを設定できません。
- `aws:elbv2:loadbalancer` - 共有 Application Load Balancer の ARN とセキュリティグループを設定します。
- `aws:elbv2:listener` - リスナールールのリストを指定して、共有 Application Load Balancer のリスナーを環境プロセスに関連付けます。

- `aws:elbv2:listenerrule` - リクエストパスに応じて、別々のプロセスにトラフィックをルーティングするリスナールールを設定します。ルールは、専用と共有の両方の Application Load Balancer に固有です。
- `aws:elasticbeanstalk:environment:process` - ヘルスチェックを設定し、環境のインスタンスで実行するプロセス用のポートとプロトコルを指定します。

Example .ebextensions/application-load-balancer-shared.config

共有 Application Load Balancer の使用を開始するには、Elastic Beanstalk コンソール、EB CLI、または API を使用してロードバランサータイプを `application` に設定し、共有ロードバランサーの使用を選択します。設定ファイルを使用して、共有ロードバランサーを設定します。

```
option_settings:
  aws:elbv2:loadbalancer:
    SharedLoadBalancer: arn:aws:elasticloadbalancing:us-
east-2:123456789012:loadbalancer/app/MySharedALB2/e574ea4c37ad2ec8
```

> **Note**
>
> このオプションは、環境の作成中にのみ設定できます。

Example .ebextensions/alb-shared-secure-listener.config

次の設定ファイルは、共有ロードバランサーのポート 443 でデフォルトの安全なリスナーを選択し、ポート 443 をリッスンするデフォルトのプロセスを設定します。

```
option_settings:
  aws:elbv2:loadbalancer:
    SharedLoadBalancer: arn:aws:elasticloadbalancing:us-
east-2:123456789012:loadbalancer/app/MySharedALB2/e574ea4c37ad2ec8
  aws:elbv2:listener:443:
    rules: default
  aws:elasticbeanstalk:environment:process:default:
    Port: '443'
    Protocol: HTTPS
```

Example .ebextensions/alb-shared-admin-rule.config

次の設定ファイルは、前の例に基づいて設定され、リクエストパスが /admin のトラフィックを、ポート 4443 でリッスンする admin という名前のプロセスにルーティングするルールを追加します。

```
option_settings:
  aws:elbv2:loadbalancer:
    SharedLoadBalancer: arn:aws:elasticloadbalancing:us-
east-2:123456789012:loadbalancer/app/MySharedALB2/e574ea4c37ad2ec8
  aws:elbv2:listener:443:
    rules: default,admin
  aws:elasticbeanstalk:environment:process:default:
    Port: '443'
    Protocol: HTTPS
  aws:elasticbeanstalk:environment:process:admin:
    HealthCheckPath: /admin
    Port: '4443'
    Protocol: HTTPS
  aws:elbv2:listenerrule:admin:
    PathPatterns: /admin/*
    Priority: 1
    Process: admin
```

Network Load Balancer の設定

ロードバランシングを有効にすると、 AWS Elastic Beanstalk 環境には Elastic Load Balancing ロードバランサーが備わり、環境内のインスタンス間でトラフィックを分散できます。Elastic Load Balancing は、いくつかのロードバランサータイプをサポートしています。それらについては、Elastic Load Balancing ユーザーガイドを参照してください。Elastic Beanstalk では、ロードバランサーを作成したり、作成した共有ロードバランサーを指定したりできます。

このトピックでは、Elastic Beanstalk が作成し、環境専用にする Network Load Balancer の設定について説明します。Elastic Beanstalk がサポートするすべてのロードバランサータイプの設定については、「Elastic Beanstalk 環境のロードバランサー」を参照してください。

> **Note**
>
> 環境の作成時にのみに環境が使用するロードバランサーのタイプを選択できます。実行中の環境のロードバランサーの行動管理の設定は変更できますが、タイプを変更することはできません。

序章

Network Load Balancer では、デフォルトのリスナーはポート 80 で TCP リクエストを受け取り、環境内のインスタンスに分散します。ヘルスチェックの動作の設定、リスナーポートの設定、または他のポートへのリスナーの追加を行うことができます。

> **Note**
>
> Classic Load Balancer や Application Load Balancer とは異なり、Network Load Balancer はアプリケーションレイヤー (レイヤー 7) の HTTP または HTTPS リスナーを配置できません。トランスポートレイヤー (レイヤー 4) TCP リスナーのみをサポートしています。HTTP および HTTPS トラフィックは、TCP 経由で環境にルーティングできます。ウェブクライアントと環境の間でセキュアな HTTPS 接続を確立するには、自己署名証明書 を環境のインスタンスにインストールして、適切なポート (通常は 443) でリッスンして HTTPS 接続を終了するようにインスタンスを設定します。設定はプラットフォームによって異なります。手順については「インスタンスでの HTTPS 終端の設定」を参照してください。次に、適切なポートでリッスンしているプロセスにマッピングされるリスナーを追加するように、Network Load Balancer を設定します。

Network Load Balancer は、アクティブなヘルスチェックをサポートしています。これらのチェックは、ルート (/) パスへのメッセージに基づきます。さらに、Network Load Balancer はパッシブなヘルスチェックもサポートしています。問題のあるバックエンドインスタンスを自動的に検出し、正常なインスタンスにのみトラフィックをルーティングします。

Elastic Beanstalk コンソールを使用した Network Load Balancer の設定

Elastic Beanstalk コンソールを使用して、環境の作成中か、後で環境の実行中に、Network Load Balancer のリスナーとプロセスを設定できます。

環境の作成中に Elastic Beanstalk コンソールで Network Load Balancer を設定するには

1. Elastic Beanstalk 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 [Environments (環境)] を選択します。
3. [新しい環境の作成] を選択して、環境の作成を開始します。
4. ウィザードのメインページで、[環境の作成] を選択する前に、[さらにオプションを設定] を選択します。
5. [高可用性] 設定プリセットを選択します。

   または、[容量] 設定カテゴリで [ロードバランサー] 環境タイプを設定します。詳細については、「容量」を参照してください。
6. [ロードバランサー] 設定カテゴリで、[編集] を選択します。
7. まだ選択されていない場合は、[Network Load Balancer] オプションを選択します。

8. Network Load Balancer の設定に、環境に必要な変更を加えます。
9. [保存] を選択し、環境に必要なその他の任意の設定変更を行います。
10. [環境を作成] を選択します。

実行中の環境の Network Load Balancer を Elastic Beanstalk コンソールで設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [ロードバランサー] 設定カテゴリで、[編集] を選択します。

   > **Note**
   >
   > [ロードバランサー] 設定カテゴリに [編集] ボタンがない場合、お客様の環境にはロードバランサーがありません。設定方法については、「環境タイプの変更」を参照してください。

5. Network Load Balancer の設定に、環境に必要な変更を加えます。
6. ページの最下部で [適用] を選択し変更を保存します。

Network Load Balancer の設定

- リスナー
- プロセス

リスナー

このリストを使用して、ロードバランサーにリスナーを指定します。各リスナーは、指定されたポートの着信クライアントトラフィックをインスタンス上のプロセスにルーティングします。初期状態では、このリストには、ポート 80 で着信トラフィックをポート 80 でリッスンする [デフォルト] という名前のプロセスにルーティングするデフォルトのリスナーが表示されます。

既存のリスナーを設定するには

1. そのテーブルエントリの横にあるチェックボックスを選択し、続いて [アクション]、[編集] の順に選択します。
2. [Network Load Balancer listener (ネットワークロードバランサーリスナー)] ダイアログボックスを使用して設定を編集し、[保存] を選択します。

リスナーを追加するには

1. [リスナーの追加] を選択します。
2. [Network Load Balancer listener (ネットワークロードバランサーリスナー)] ダイアログボックスで必要な設定を行い、[追加] を選択します。

[Network Load Balancer listener (ネットワークロードバランサーリスナー)] ダイアログボックスを使用して、リスナーがトラフィックをリッスンするポートを設定し、トラフィックをルーティングするプロセス (プロセスがリッスンするポートによって指定された) を選択します。

プロセス

このリストを使用して、ロードバランサーにプロセスを指定します。プロセスは、トラフィックをルートするターゲットです。各リスナーは、指定されたポートの着信クライアントトラフィックをインスタンス上のプロセスにルーティングします。初期状態では、このリストには、ポート 80 で着信トラフィックをリッスンするデフォルトのプロセスが表示されます。

既存のプロセスの設定を編集するか、あるいは新しいプロセスを追加します。リストのプロセスの編集またはプロセスの追加を開始するには、リスナーリストと同じステップを使用します。[環境プロセス] ダイアログボックスが開きます。

Network Load Balancer の環境プロセスダイアログボックスでの設定

- 定義
- ヘルスチェック

定義

これらの設定を使用して、[名前] およびリクエストをリッスンする [プロセスポート] でプロセスを定義します。

ヘルスチェック

次の設定を使用して、プロセスのヘルスチェックを設定します。

- [Interval (間隔)] - 個々のインスタンスのヘルスチェックの間隔 (秒単位)。
- [Healthy threshold (正常のしきい値)] - Elastic Load Balancing がインスタンスのヘルス状態を変更する前に、ヘルスチェックに合格しなければならない回数 (Network Load Balancer で、[Unhealthy threshold (非正常のしきい値)] は読み取り専用の設定であり、正常のしきい値と常に等しくなります)。
- [Deregistration delay (登録解除の遅延)] - インスタンスの登録を解除する前にアクティブリクエストの完了を待機する時間 (秒単位)。

> **Note**
>
> Elastic Load Balancing ヘルスチェックは、環境の Auto Scaling グループのヘルスチェック動作に影響しません。Elastic Load Balancing ヘルスチェックに失敗したインスタンスは、Amazon EC2 Auto Scaling を手動で設定していなければ、Amazon EC2 Auto Scaling によって自動的に置き換えられません。詳細については、「Elastic Beanstalk 環境の Auto Scaling ヘルスチェック設定」を参照してください。

ヘルスチェックと、それが環境の全体的な状態に与える影響の詳細については、「ベーシックヘルスレポート」を参照してください。

例: エンドツーエンド暗号化を使用する環境の Network Load Balancer

この例では、アプリケーションはエンドツーエンドのトラフィック暗号化を必要とします。これらの要件を満たすように環境の Network Load Balancer を設定するには、ポート 443 をリッスンするようにデフォルトのプロセスを設定し、トラフィックをデフォルトのプロセスにルーティングするリスナーをポート 443 に追加して、デフォルトのリスナーを無効にします。

この例でロードバランサーを設定するには

1. デフォルトプロセスを設定します。デフォルトのプロセスを選択し、[アクション] で [編集] を選択します。[プロセスポート] に「443」と入力します。

2. ポート 443 リスナーを追加します。新しいリスナーを追加します。[リスナーポート] で 443 と入力します。[プロセスポート] として 443 が選択されていることを確認します。

   これでリストに追加のリスナーが表示されます。

3. デフォルトのポート 80 リスナーを無効にします。デフォルトのリスナーについて、[有効] オプションをオフにします。

EB CLI を使用した Network Load Balancer の設定

eb create の実行時に、EB CLI によりロードバランサータイプの選択が求められます。

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
1) classic
2) application
3) network
```

```
(default is 1): 3
```

`--elb-type` オプションでロードバランサータイプを指定することもできます。

```
$ eb create test-env --elb-type network
```

Network Load Balancer の名前空間

Network Load Balancer に関連する設定は、以下の名前空間にあります。

- `aws:elasticbeanstalk:environment` - 環境のロードバランサーのタイプを選択します。Network Load Balancer の値は `network` です。
- `aws:elbv2:listener` - Network Load Balancer のリスナーを設定します。これらの設定は、Classic Load Balancer に対する `aws:elb:listener` の設定にマッピングされます。
- `aws:elasticbeanstalk:environment:process` - ヘルスチェックを設定し、環境のインスタンスで実行するプロセス用のポートとプロトコルを指定します。ポートとプロトコル設定は Classic Load Balancer のリスナー用の `aws:elb:listener` のインスタンスポートおよびインスタンスプロトコル設定にマッピングされます。ヘルスチェックの設定は、`aws:elb:healthcheck` および `aws:elasticbeanstalk:application` 名前空間の設定にマッピングされます。

Example .ebextensions/network-load-balancer.config

Network Load Balancer の使用を開始するには、設定ファイルを使用してロードバランサータイプを `network` に設定します。

```
option_settings:
  aws:elasticbeanstalk:environment:
    LoadBalancerType: network
```

> **Note**
>
> 環境の作成中にのみ、ロードバランサータイプを設定できます。

Example .ebextensions/nlb-default-process.config

以下の設定ファイルでは、デフォルトプロセスのヘルスチェックの設定を変更します。

```
option_settings:
  aws:elasticbeanstalk:environment:process:default:
    DeregistrationDelay: '20'
    HealthCheckInterval: '10'
    HealthyThresholdCount: '5'
    UnhealthyThresholdCount: '5'
    Port: '80'
    Protocol: TCP
```

Example .ebextensions/nlb-secure-listener.config

以下の設定ファイルでは、ポート 443 のセキュアなトラフィックのリスナーと、ポート 443 をリッスンする一致するターゲットプロセスが追加されます。

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
```

この `DefaultProcess` オプションは、Application Load Balancer によりこのように名前が付けられており、特定のパスのトラフィックの同一ポートのデフォルトでないリスナーを設定することができます (詳細については Application Load Balancer を参照)。Network Load Balancer の場合、このオプションでは、このリスナーの唯一のターゲットプロセスを指定します。

この例では、セキュアな (HTTPS) トラフィックをリッスンするため、プロセスは `https` という名前になります。Network Load Balancer は TCP でのみ使用できるため、リスナーは、TCP プロトコルを使用して、指定ポートのプロセスにトラフィックを送信します。HTTP および HTTPS のネットワークトラフィックは TCP の上で実装されるため、これは問題ありません。

アクセスログの設定

設定ファイルを使用して、Amazon S3 バケットにアクセスログをアップロードするよう環境のロードバランサーを設定できます。手順については、GitHub の次の例の設定ファイルを参照してください。

- `loadbalancer-accesslogs-existingbucket.config` - 既存の Amazon S3 バケットにアクセスログをアップロードするようロードバランサーを設定します。

- `loadbalancer-accesslogs-newbucket.config` - 新しいバケットにアクセスログをアップロードするようロードバランサーを設定します。

Elastic Beanstalk 環境にデータベースを追加する

Elastic Beanstalk は、Amazon Relational Database Service (Amazon RDS) とのデータベース統合を提供します。Elastic Beanstalk を使用して、MySQL、PostgreSQL、Oracle、または SQL Server データベースを既存または新規の環境に作成時に追加できます。環境に結合されたデータベースインスタンスを追加すると、Elastic Beanstalk は接続情報をアプリケーションに提供します。これは、データベースのホスト名、ポート、ユーザー名、パスワード、およびデータベース名の環境プロパティを設定します。

結合データベースを使用する利点

アプリケーションでデータベースインスタンスを使用したことがない場合は、最初に、このトピックで説明するプロセスを使用して、Elastic Beanstalk サービスを使用してテスト環境にデータベースを追加することをお勧めします。これにより、 Elastic Beanstalk 外部のデータベースに必要な追加の設定作業をせずに、アプリケーションで環境プロパティを読み取り、接続文字列を作成し、データベースインスタンスに接続できることを確認できます。

本番環境に移行する際の考慮事項

アプリケーションがデータベースで正しく動作することを確認したら、本番環境に移行することを検討できます。この時点で、Elastic Beanstalk 環境からデータベースをデカップリングして、柔軟性が高い設定に移行することもできます。デカップリングされたデータベースは、外部 Amazon RDS データベースインスタンスとして動作し続けることができます。環境の健全性は、データベースをデカップリングしても影響を受けません。環境を終了する必要がある場合は、その環境を終了し、Elastic Beanstalk の外部で引き続きデータベースが使用可能な状態で運用することもできます。

分離されたデータベースに移行する利点

外部データベースの使用にはいくつかの利点があります。複数の環境から外部データベースに接続したり、統合データベースでサポートされていないデータベースタイプを使用したり、Blue-Green デプロイメントを実行したりできます。Elastic Beanstalk が作成したデカップリングされたデータベースを使用する代わりに、Elastic Beanstalk 環境の外部でデータベースインスタンスを作成することもできます。どちらの方法でも、Elastic Beanstalk 環境外部のデータベースインスタンスを使用することになり、追加のセキュリティグループと接続文字列の設定が必要になります。詳細については、「Amazon RDS で Elastic Beanstalk を使用する」を参照してください。

セクション

- データベースのライフサイクル
- コンソールを使用して環境に Amazon RDS DB インスタンスを追加する
- データベースに接続
- コンソールを使用した統合 RDS DB インスタンスの設定
- 設定ファイルを使用した統合 RDS DB インスタンスの設定
- コンソールを使用した RDS DB インスタンスのデカップリング
- 設定ファイルを使用して RDS DB インスタンスをデカップリングする

データベースのライフサイクル

Elastic Beanstalk 環境からデータベースをデカップリングした後のデータベースの状態を選択できます。選択できるオプションは、総称して「削除ポリシー」と呼ばれます。次の削除ポリシーは、Elastic Beanstalk 環境からデカップリングした後、または Elastic Beanstalk 環境を終了した後のデータベースに適用されます。

- Snapshot (スナップショット) – Elastic Beanstalk はデータベースを終了する前に、データベースのスナップショットを保存します。DB インスタンスを Elastic Beanstalk 環境に追加するとき、またはスタンドアロンデータベースを作成するときに、スナップショットからデータベースを復元できます。スナップショットからの新しいスタンドアロン DB インスタンスの作成の詳細については、Amazon RDS ユーザーガイドの「DB スナップショットからの復元」を参照してください。データベースのスナップショットを保存するときに料金が発生する場合があります。詳細については、Amazon RDS 料金表の「Backup ストレージ」セクションを参照してください。
- Delete (削除) – Elastic Beanstalk はデータベースを終了します。終了後、データベースインスタンスはどのオペレーションにも使用できなくなります。
- Retain (保持) – データベースインスタンスは終了しません。Elastic Beanstalk からデカップリングされていますが、引き続き利用可能で操作可能です。その後、外部 Amazon RDS データベースインスタンスとしてデータベースに接続するように、1 つまたは複数の環境を設定できます。詳細については、「Amazon RDS で Elastic Beanstalk を使用する」を参照してください。

コンソールを使用して環境に Amazon RDS DB インスタンスを追加する

Elastic Beanstalk コンソールを使用して、DB インスタンスを環境に追加できます。

お客様の環境に DB インスタンスを追加するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. DB エンジンを選択して、ユーザー名とパスワードを入力します。
6. ページの最下部で [適用] を選択し変更を保存します。

次のオプションを設定できます。

- [スナップショット] – 既存のデータベーススナップショットを選択します。Elastic Beanstalk はスナップショットを復元し、環境に追加します。デフォルト値は [なし] です。デフォルト値が [None] (なし) の場合、このページの他の設定を使用して新しいデータベースを設定できます。
- [エンジン] – データベースエンジンを選択します。
- [エンジンバージョン] – データベースエンジンの特定のバージョンを選択します。
- [インスタンスクラス] – DB インスタンスのクラスを選択します。DB インスタンスクラスの詳細については、https://aws.amazon.com/rds/ を参照してください。
- [ストレージ] – データベースのためにプロビジョニングするストレージの容量を選択します。割り当て済みストレージを後で増やすことはできますが、減らすことはできません。ストレージの割り当ての詳細については、「Features」を参照してください。
- Username (ユーザー名) – 数字と文字のみの組み合わせを使用して、任意のユーザー名を入力します。
- [パスワード] – 8 ~ 16 文字の印刷可能な ASCII 文字 (/、\、@ は除く) を含む任意のパスワードを入力します。
- [可用性] – 高可用性のために、2 番目のアベイラビリティーゾーンでウォームバックアップを実行するには、[高 (マルチ AZ)] を選択します。
- Database deletion policy (データベース削除ポリシー) – 削除ポリシーで、環境からデカップリングされた後のデータベースの状態が決まります。これに設定できる値は、`Create Snapshot`、`Retain`、または `Delete` です。これらの値については、このトピックの中の「データベースのライフサイクル」を参照してください。

> **Note**
>
> Elastic Beanstalk を使用して、データベースのマスターユーザーを作成し、ユーザー名とパスワードを指定します。マスターユーザーとその権限の詳細については、「マスターユーザーアカウント特権」を参照してください。

DB インスタンスの追加には約 10 分かかります。更新が完了すると、新しいデータベースは環境にカップリングされます。DB インスタンスのホスト名とその他の接続情報は以下の環境プロパティを通じてアプリケーションが使用できるようになります。

プロパティ名	説明	プロパティ値
`RDS_HOSTNAME`	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
`RDS_PORT`	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。
`RDS_DB_NAME`	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
`RDS_USERNAME`	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。
`RDS_PASSWORD`	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

データベースに接続

アプリケーションの内部から環境変数を使用してデータベースに接続する場合は、接続情報を利用します。アプリケーションで Amazon RDS を使用する方法については、以下のトピックを参照してください。

- Java SE – データベースへの接続 (Java SE プラットフォーム)
- Java と Tomcat – データベースへの接続 (Tomcat プラットフォーム)
- Node.js – データベースへの接続
- .NET – データベースへの接続
- PHP – PDO または MySQLi を使用してデータベースに接続
- Python – データベースへの接続
- Ruby – データベースへの接続

コンソールを使用した統合 RDS DB インスタンスの設定

[Elastic Beanstalk console] (Elastic Beanstalk コンソール) から、環境の [Configuration] (設定) ページの [Database] (データベース) セクションで、データベースインスタンスの設定を表示および変更できます。

Elastic Beanstalk コンソールで環境の DB インスタンスを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。

データベースの作成後、[Instance class] (インスタンスクラス)、[Storage] (ストレージ)、[Password] (パスワード)、[Availability] (可用性)、および [Database deletion policy] (データベース削除ポリシー) の設定を変更することができます。インスタンスクラスを変更すると、Elastic Beanstalk は DB インスタンスを再プロビジョニングします。

データベースを環境に関連付けるのに Elastic Beanstalk が不要になった場合は、[Decouple database] (データベースをデカップリング) を選択して、データベースをデカップリングすることを選択できます。このオペレーションに関連するオプションと考慮事項を理解することが重要です。

詳細については、「the section called “コンソールを使用した RDS DB インスタンスのデカップリング”」を参照してください。

> **警告**
>
> Elastic Beanstalk で提供される機能以外 (例えば、Amazon RDS コンソールの機能など) で、カップリングされたデータベースインスタンスの設定を変更しないでください。これを行うと、Amazon RDS の DB 設定が環境定義と同期しない可能性があります。環境を更新または再起動すると、環境で指定された設定は、Elastic Beanstalk 以外の設定を上書きします。Elastic Beanstalk が直接サポートしていない設定を変更する必要がある場合は、Elastic Beanstalk の設定ファイルを使用してください。

設定ファイルを使用した統合 RDS DB インスタンスの設定

設定ファイルを使用して、環境のデータベースインスタンスを設定できます。`aws:rds:dbinstance` 名前空間のオプションを使用します。次の例では、割り当てられたデータベースストレージサイズを 100 GB に変更します。

Example .ebextensions/db-instance-options.config

```
option_settings:
  aws:rds:dbinstance:
    DBAllocatedStorage: 100
```

Elastic Beanstalk がサポートしていない DB インスタンスのプロパティを設定する場合は、やはり設定ファイルを使用し、`resources` キーを使用して設定を指定することができます。次の例では、`StorageType` および `Iops` Amazon RDS のプロパティに値を設定します。

Example .ebextensions/db-instance-properties.config

```
Resources:
  AWSEBRDSDatabase:
    Type: AWS::RDS::DBInstance
    Properties:
      StorageType:io1
      Iops: 1000
```

コンソールを使用した RDS DB インスタンスのデカップリング

環境の健全性に影響を与えることなく、データベースを Elastic Beanstalk 環境からデカップリングできます。データベースをデカップリングする前に、次の要件を考慮してください。

- データベースがデカップリングされた後のデータベースの状態

  データベースのスナップショットを作成して終了するか、データベースを保持して Elastic Beanstalk 外部のスタンドアロンデータベースとして動作させるか、データベースを完全に削除するかを選択できます。[Database deletion policy] (データベース削除ポリシー) 設定により、この結果が決まります。削除ポリシーの詳細については、このトピックの中の「データベースのライフサイクル」を参照してください。

- デカップリングする前に、データベース設定を変更する必要があるか?

  データベースの設定を変更する必要がある場合は、データベースをデカップリングする前に適用する必要があります。これには、[Database deletion policy] (データベース削除ポリシー) の変更も含まれます。[Decouple database] (データベースをデカップリング) 設定と同時に送信される保留中の変更は無視され、デカップリング設定のみが適用されます。

環境から DB インスタンスをデカップリングするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. [Database settings] (データベース設定) セクションのすべての設定値を確認します。特に、[Database deletion policy] (データベース削除ポリシー) は、デカップリング後のデータベースの状態を決めるので、よく確認します。

   その他の設定がすべて正しい場合は、ステップ 6 に進み、データベースをデカップリングします。

> **Warning**
>
> [Decouple database] (データベースをデカップリング) とは別に、[Database deletion policy] (データベース削除ポリシー) 設定を適用することが重要です。[Decouple database] (データベースをデカップリング) と新たに選択した [Database deletion policy] (データベース削除ポリシー) の両方を保存しようとして [Apply] (適用) をクリックしても、選択した新しい削除ポリシーは無視されます。Elastic Beanstalk は、以前に設定された削除ポリシーに従ってデータベースをデカップリングします。以前に設定された削除ポリシーが `Delete` または `Create Snapshot` の場合、適用しようとした保留中のポリシーは適用されず、データベースが失われる危険性があります。

いずれかの設定で更新が必要な場合は、次の操作を行います。

1. [Database settings] (データベース設定) パネルで必要な修正を行います。
2. [Apply] (適用) を選択します。データベースの設定の変更を保存するには、数分かかります。
3. ステップ 3 に戻り、ナビゲーションペインから [Configuration] (設定) を選択します。

6. ペインの [Database connection] (データベース接続) セクションに移動します。

7. [Decouple database] (データベースをデカップリング) を選択します。
8. [Apply] (適用) をクリックして、データベースのデカップリング操作を開始します。

削除ポリシー設定によって、データベースがどうなるかと、データベースをデカップリングするために必要な時間の長さが決まります。

- 削除ポリシーが `Delete` に設定されている場合、データベースが削除されます。データベースのサイズによっては、オペレーションにおよそ 10 ~ 20 分かかることがあります。
- 削除ポリシーが `Snapshot` に設定されている場合、データベースのスナップショットが作成されます。その後、データベースが削除されます。このプロセスに必要な時間は、データベースのサイズによって異なります。
- 削除ポリシーが `Retain` に設定されている場合、データベースは引き続き Elastic Beanstalk 環境の外部で動作します。通常、データベースをデカップリングするには 5 分もかかりません。

Elastic Beanstalk 環境の外部にデータベースを保持することに決めた場合は、追加の手順を実行してデータベースを設定する必要があります。詳細については、「Amazon RDS で Elastic Beanstalk を使用する」を参照してください。デカップリングしたデータベースを実稼働環境で使用する場合は、データベースが使用するストレージタイプがワークロードに適していることを確認します。詳細については、Amazon RDS ユーザーガイドの「Amazon RDS DB インスタンスストレージ」および「Amazon RDS DB インスタンスを変更する」を参照してください。

設定ファイルを使用して RDS DB インスタンスをデカップリングする

環境の健全性に影響を与えずに、DB インスタンスを Elastic Beanstalk 環境からデカップリングできます。データベースがデカップリングされたとき、データベースインスタンスは、適用された [データベース削除ポリシー] に従います。

データベースをデカップリングするために必要なオプションは両方とも、the section called “aws:rds:dbinstance” 名前空間にあります。その内容は次のとおりです。

- `DBDeletionPolicy` オプションで、削除ポリシーが設定されます。これに設定できる値は、`Snapshot`、`Delete`、または `Retain` です。これらの値については、このトピックの中の「データベースのライフサイクル」を参照してください。
- `HasCoupledDatabase` オプションで、環境にカップリングされたデータベースがあるかどうかが決まります。
  - `true` にした場合、Elastic Beanstalk によって環境にカップリングされた新しい DB インスタンスが作成されます。
  - `false` にした場合、Elastic Beanstalk によって環境から DB インスタンスがデカップリングされます。

デカップリングする前にデータベース設定を変更する場合は、まず設定の変更を別のオペレーションで適用します。これには、`DBDeletionPolicy` 設定の変更も含まれます。変更が適用されたら、別のコマンドを実行してデカップリングオプションを設定します。他の設定とデカップリング設定を同時に送信すると、デカップリング設定は適用されますが、他のオプション設定は無視されます。

> **Warning**
>
> `DBDeletionPolicy` と `HasCoupledDatabase` の設定を 2 つの別々のオペレーションとして適用するようにコマンドを実行することが重要です。現在有効な削除ポリシーが `Delete` または `Snapshot` に設定されてしまっている場合、データベースが失われる危険性がありま

す。データベースは、適用しようとした保留中の削除ポリシーではなく、現在有効な削除ポリシーに従います。

環境から DB インスタンスをデカップリングするには

次の手順に従って、Elastic Beanstalk 環境からデータベースをデカップリングします。EB CLI または を使用してステップ AWS CLI を完了できます。詳細については、「設定ファイルによる高度な環境のカスタマイズ」を参照してください。

1. 削除ポリシーを変更する場合は、次の形式で設定ファイルを設定します。この例では、削除ポリシーは保持に設定されています。

   Example

   ```
   option_settings:
     aws:rds:dbinstance:
       DBDeletionPolicy: Retain
   ```

2. 任意のツールを使用してコマンドを実行し、設定の更新を完了します。
3. 設定ファイルをセットアップして、`HasCoupledDatabase` を `false` に設定します。

   Example

   ```
   option_settings:
     aws:rds:dbinstance:
       HasCoupledDatabase: false
   ```

4. 任意のツールを使用してコマンドを実行し、設定の更新を完了します。

削除ポリシー設定によって、データベースがどうなるかと、データベースを分離するために必要な時間の長さが決まります。

- 削除ポリシーが `Delete` に設定されている場合、データベースが削除されます。データベースのサイズによっては、オペレーションにおよそ 10 ~ 20 分かかることがあります。
- 削除ポリシーが `Snapshot` に設定されている場合、データベースのスナップショットが作成されます。その後、データベースが削除されます。このプロセスに必要な時間は、データベースのサイズによって異なります。

- 削除ポリシーが `Retain` に設定されている場合、データベースは引き続き Elastic Beanstalk 環境の外部で動作します。通常、データベースをデカップリングするには 5 分もかかりません。

Elastic Beanstalk 環境の外部にデータベースを保持することに決めた場合は、追加の手順を実行してデータベースを設定する必要があります。詳細については、「Amazon RDS で Elastic Beanstalk を使用する」を参照してください。デカップリングしたデータベースを実稼働環境で使用する場合は、データベースが使用するストレージタイプがワークロードに適していることを確認します。詳細については、Amazon RDS ユーザーガイドの「Amazon RDS DB インスタンスストレージ」および「Amazon RDS DB インスタンスを変更する」を参照してください。

AWS Elastic Beanstalk 環境セキュリティ

Elastic Beanstalk には、環境と、その中にある Amazon EC2 インスタンスのサービスアクセス (セキュリティ) を制御するいくつかのオプションが用意されています。このトピックでは、これらのオプションの設定について説明します。

セクション

- 環境セキュリティの設定
- 環境セキュリティ設定の名前空間

環境セキュリティの設定

Elastic Beanstalk の環境セキュリティ設定は、Elastic Beanstalk コンソールで変更できます。

Elastic Beanstalk コンソールで環境のサービスアクセス (セキュリティ) を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [Service access] (サービスアクセス) 設定カテゴリで、[Edit] (編集) を選択します。

以下の設定を使用できます。

設定

- サービスロール
- EC2 キーペア

- IAM インスタンスプロファイル

サービスロール

サービスロールを選択して、お使いの Elastic Beanstalk environment 環境に関連付けます。Elastic Beanstalk は、ユーザーに代わって他のサービスにアクセスするときに AWS サービスロールを引き受けます。詳細については、「Elastic Beanstalk サービスロールの管理」を参照してください。

EC2 キーペア

Elastic Beanstalk アプリケーション用にプロビジョンされた Amazon Elastic Compute Cloud (Amazon EC2)インスタンスには、Amazon EC2 キーペアを使用して安全にログインできます。キーペアを作成する手順については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 を使用したキーペアの作成」を参照してください。

> **Note**
>
> キーペアを作成すると、Amazon EC2 はパブリックキーのコピーを保存します。環境インスタンスに接続するためにこれを使用する必要がなくなった場合は、Amazon EC2 から削除できます。詳細については、「Amazon EC2 ユーザーガイド」の「キーペアの削除」を参照してください。

ドロップダウンメニューから適切な [EC2 key pair] を選択し、環境のインスタンスに割り当てます。キーペアを割り当てるとパブリックキーがインスタンスに保存され、ローカルに保存されているプライベートキーの認証が行われます。プライベートキーは保存されません AWS。

Amazon EC2 インスタンスに接続する方法の詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスへの接続」と「PuTTY を使用した Windows から Linux/UNIX インスタンスへの接続」を参照してください。

IAM インスタンスプロファイル

EC2 のインスタンスプロファイルは、Elastic Beanstalk 環境で起動するインスタンスに適用される IAM ロールです。Amazon EC2 インスタンスは、Amazon S3 にログをアップロードするなど、 へのリクエストに署名 AWS して APIs にアクセスするインスタンスプロファイルロールを引き受けます。

Elastic Beanstalk コンソールで初めて環境を作成する場合、Elastic Beanstalk により、デフォルトのアクセス許可のセットでインスタンスプロファイルを作成するように求められます。このプロファイルにアクセス許可を追加して、インスタンスに他の AWS サービスへのアクセスを提供できます。詳細については、「Elastic Beanstalk インスタンスプロファイルの管理」を参照してください。

> **Note**
>
> 以前は、Elastic Beanstalk は、 AWS アカウント`aws-elasticbeanstalk-ec2-role`が初めて環境を作成したときに、 という名前のデフォルトの EC2 インスタンスプロファイルを作成していました。このインスタンスプロファイルには、デフォルトの管理ポリシーが含まれていました。アカウントに既にこのインスタンスプロファイルがある場合、引き続き環境に割り当てることができます。
> ただし、最近の AWS セキュリティガイドラインでは、 AWS サービスが信頼ポリシーを使用して他の AWS サービスである EC2 へのロールを自動的に作成することはできません。これらのセキュリティガイドラインにより、Elastic Beanstalk はデフォルトの `aws-elasticbeanstalk-ec2-role` インスタンスプロファイルを作成しなくなりました。

> **Note**
>
> EC2 インスタンスのセキュリティには、EC2 インスタンスのファイアウォールルールを指定する別の側面があります。これは EC2 セキュリティグループによって制御されます。詳細については、「お客様のElastic Beanstalk 環境に対する Amazon EC2 インスタンス」を参照してください。

環境セキュリティ設定の名前空間

Elastic Beanstalk は、環境のセキュリティをカスタマイズできるように、次の名前空間に設定オプションを提供します。

- `aws:elasticbeanstalk:environment` – `ServiceRole` オプションを使用して環境のサービスロールを設定します。
- `aws:autoscaling:launchconfiguration` – `EC2KeyName`、、`IamInstanceProfile`、および `SecurityGroups`オプションを使用して`DisableDefaultEC2SecurityGroup`、環境の Amazon EC2 インスタンスのアクセス許可を設定します。

EB CLI および Elastic Beanstalk コンソールでは、上記のオプションに推奨値が適用されます。設定ファイルを使用して同じファイルを設定する場合は、これらの設定を削除する必要があります。詳細については、「推奨値」を参照してください。

Elastic Beanstalk 環境でのリソースのタグ付け

AWS Elastic Beanstalk 環境にタグを適用できます。タグは、 AWS リソースに関連付けられたキーと値のペアです。Elastic Beanstalk リソースのタグ付け、ユースケース、タグのキーと値の制約、サポートされているリソースタイプの詳細については、「Elastic Beanstalk アプリケーションリソースのタグ付け」を参照してください。

Elastic Beanstalk は、環境リソース自体、および Elastic Beanstalk が環境用に作成する他の AWS リソースに環境タグを適用します。タグを使用して、環境内の特定のリソースレベルでアクセス権限を管理することができます。詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 リソースのタグ付け」を参照してください。

Elastic Beanstalk では、デフォルトで環境にいくつかのタグが適用されます。

- `elasticbeanstalk:environment-name` – 環境の名前。
- `elasticbeanstalk:environment-id` – 環境 ID。
- `Name` – こちらも環境の名前です。`Name` は、Amazon EC2 ダッシュボードでリソースを識別およびソートするために使用されます。

これらのデフォルトのタグは編集できません。

タグは、Elastic Beanstal 環境の作成時に指定できます。既存の環境では、タグの追加や削除、既存タグの値の更新ができます。環境には、デフォルトのタグを含む最大 50 個のタグを含めることができます。

環境の作成時にタグを追加する

Elastic Beanstalk コンソールを使用して環境を作成する場合、新しい環境の作成ウィザードの [Modify tags] (タグの変更) 設定ページでタグのキーと値を指定できます。

EB CLI を使用して環境を作成する場合は、eb create で `--tags` オプションを使用してタグを追加します。

```
~/workspace/my-app$ eb create --tags mytag1=value1,mytag2=value2
```

AWS CLI または他の API ベースのクライアントでは、 create-environment コマンドで `--tags`パラメータを使用します。

```
$ aws elasticbeanstalk create-environment \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --application-name my-app --environment-name my-env --cname-prefix my-app --
version-label v1 --template-name my-saved-config
```

保存済み設定にはユーザー定義タグが含まれます。環境の作成時にタグが含まれる情報を適用すると、新しいタグを指定しない限り、これらのタグが新しい環境に適用されます。前述の方法の 1 つを使用して環境にタグを追加した場合、保存済み設定に定義されているタグはすべて破棄されます。

既存環境のタグの管理

既存の Elastic Beanstalk 環境のタグを追加、更新、削除できます。Elastic Beanstalk により、環境のリソースに変更が適用されます。

ただし、Elastic Beanstalk によって環境に適用されるデフォルトのタグは編集できません。

Elastic Beanstalk コンソールで環境のタグを管理するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[Tags] を選択します。

   タグの管理ページに、現在環境に存在するタグのリストが表示されます。

4. タグを追加、更新、または削除します。
   - タグを追加するには、リストの下部にある空のボックスに入力します。別のタグを追加するには、[タグの追加] を選択すると、Elastic Beanstalk によって空ボックスのペアが追加されます。
   - タグのキーまたは値を更新するには、タグの行で対応するボックスを編集します。
   - タグを削除するには、タグの値ボックスの横にある [Remove (削除)] を選択します。
5. ページの最下部で [適用] を選択し変更を保存します。

EB CLI を使用して環境を更新する場合は、eb tags を使用してタグを追加、更新、削除、一覧表示します。

たとえば、次のコマンドでは、デフォルト環境のタグを一覧表示します。

```
~/workspace/my-app$ eb tags --list
```

次のコマンドでは、mytag1 タグを更新して mytag2 タグを削除します。

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2
```

オプションの完全なリストおよび詳細な例については、「eb tags」を参照してください。

AWS CLI または他の API ベースのクライアントでは、 list-tags-for-resource コマンドを使用して環境のタグを一覧表示します。

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:environment/my-app/my-env"
```

環境のタグを追加、更新、または削除するには、update-tags-for-resource コマンドを使用します。

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:environment/my-
app/my-env"
```

追加するタグと更新するタグを update-tags-for-resource の `--tags-to-add` パラメータで指定します。存在していないタグが追加され、既存のタグの値が更新されます。

> **Note**
>
> Elastic Beanstalk 環境でこれら 2 つの AWS CLI コマンドを使用するには、環境の ARN が必要です。ARN を取得するには、次のコマンドを使用します。
>
> ```
> $ aws elasticbeanstalk describe-environments
> ```

環境変数およびその他のソフトウェア設定

[更新、モニタリング、ロギングの設定] ページでは、アプリケーションを実行する Amazon Elastic Compute Cloud (Amazon EC2) インスタンス上のソフトウェアを設定できます。環境変数、 AWS X-Ray デバッグ、インスタンスログの保存とストリーミング、プラットフォーム固有の設定を設定できます。

トピック

- プラットフォーム固有の設定を構成する
- 環境プロパティ (環境変数) の設定
- ソフトウェア設定の名前空間
- 環境プロパティへのアクセス
- AWS X-Ray デバッグの設定
- Elastic Beanstalk 環境ログを表示する

プラットフォーム固有の設定を構成する

すべての環境で利用できる標準のオプションセットに加え、ほとんどの Elastic Beanstalk プラットフォームでは、言語固有またはフレームワーク固有の指定を行うことができます。これらは、[更新、モニタリング、およびロギングの設定] ページの [プラットフォームソフトウェア] セクションに表示され、次の形式を取ることができます。

- プリセットの環境プロパティ – Ruby プラットフォームでは、`RACK_ENV` および `BUNDLE_WITHOUT` のようなフレームワーク設定の環境プロパティを使用します。
- プレースホルダーの環境プロパティ – Tomcat プラットフォームでは、どのような値にも設定されない `JDBC_CONNECTION_STRING` という名前の環境プロパティを定義します。この設定タイプは、旧式のプラットフォームバージョンで一般的でした。
- 設定オプション – ほとんどのプラットフォームでは、`aws:elasticbeanstalk:xray` や `aws:elasticbeanstalk:container:python` などのプラットフォーム固有または共有の名前空間で設定オプションを定義します。

Elastic Beanstalk コンソールでプラットフォーム固有の設定を指定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。

3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [プラットフォームソフトウェア] で、必要なオプション設定を変更します。
6. ページの最下部で [適用] を選択し変更を保存します。

プラットフォーム固有のオプションと、コードで環境プロパティ値を取得する方法の詳細については、該当する言語やフレームワークのプラットフォームトピックを参照してください。

- Docker – the section called “環境設定”
- Go – Elastic Beanstalk Go プラットフォームを使用する
- Java SE – Elastic Beanstalk Java SE プラットフォームの使用
- Tomcat – Elastic Beanstalk Tomcat プラットフォームを使用する
- .NET Core on Linux – Elastic Beanstalk .NET Core on Linux プラットフォームの使用
- .NET – Elastic Beanstalk .NET Windows プラットフォームの使用
- Node.js – Elastic Beanstalk Node.js プラットフォームを使用する
- PHP – Elastic Beanstalk PHP プラットフォームを使用する
- Python – Elastic Beanstalk Python プラットフォームを使用する
- Ruby – Elastic Beanstalk Ruby プラットフォームを使用する

環境プロパティ (環境変数) の設定

環境プロパティ (環境変数とも呼ばれます) を使用して、エンドポイント、デバッグ設定、その他の情報をアプリケーションに渡すことができます。環境変数は、開発、テスト、ステージング、本番稼働など、さまざまな目的で複数の環境でアプリケーションを実行するのに役立ちます。

さらに、データベースを環境に追加すると、Elastic Beanstalk は などの環境変数を設定し`RDS_HOSTNAME`、アプリケーションコードで読み取って接続オブジェクトまたは文字列を構築できます。

Elastic Beanstalk コンソールで環境変数を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。

4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. Runtime 環境変数まで下にスクロールします。
6. 環境変数の追加 を選択します。
7. ソース プレーンテキストを選択します。

> **Note**
>
> ドロップダウンの Secrets Manager と SSM パラメータストアの値は、認証情報や API キーなどの機密データを保存するシークレットとして環境変数を設定するためのものです。詳細については、「AWS Secrets Manager および Parameter Store での Elastic Beanstalk AWS Systems Manager の使用」を参照してください。

8. 環境変数名と環境変数値のペアを入力します。
9. さらに変数を追加する必要がある場合は、ステップ 6 からステップ 8 を繰り返します。
10. ページの最下部で [適用] を選択し変更を保存します。

環境プロパティの制限

- キーには、英数字および記号 `_ . : / + \ - @` を含めることができます。

  上記の記号は、環境プロパティーキーには使用できますが、環境のプラットフォームの環境変数名には使用できない場合があります。すべてのプラットフォームとの互換性を考慮して、環境プロパティには次のパターン (`[A-Z_][A-Z0-9_]*`) のみ使用してください。
- 値には、英数字、空白、および記号 `_ . : / = + \ - @ ' "` を含めることができます。

> **Note**
>
> 環境プロパティ値の一部の文字で囲う値はエスケープする必要があります。バックスラッシュ文字 (\) を使用して、いくつかの特殊文字と制御文字を表します。次のリストには、エスケープする必要があるいくつかの文字を表す例が含まれています。
>
> - バックスラッシュ (\) — `\\` の使用を表す
> - 一重引用符 (') — `\'` の使用を表す
> - 二重引用符 (") — `\"` の使用を表す

- キーと値は大文字と小文字が区別されます。

- *key*=*value* の形式の文字列として保存されている場合は、すべての環境プロパティの合計サイズが 4,096 バイトを超えることはできません。

ソフトウェア設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクをパフォーマンスできます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

Elastic Beanstalk 設定ファイルを使用して、ソースコードで環境プロパティと設定オプションを設定できます。`aws:elasticbeanstalk:application:environment`名前空間を使用して、環境プロパティを定義します。

Example .ebextensions/options.config

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    API_ENDPOINT: www.example.com/api
```

設定ファイルまたは AWS CloudFormation テンプレートを使用してカスタムリソースを作成する場合は、 関数を使用して AWS CloudFormation リソースに関する情報を取得し、デプロイ中に環境プロパティに動的に割り当てることができます。以下、GitHub リポジトリの elastic-beanstalk-samples の例では、Ref 関数を使用して、作成される Amazon SNS トピックの ARN を取得し、`NOTIFICATION_TOPIC` という名前の環境プロパティに割り当てています。

> **メモ**
>
> - AWS CloudFormation 関数を使用して環境プロパティを定義する場合、関数が評価される前に Elastic Beanstalk コンソールにプロパティの値が表示されます。`get-config` プラットフォームスクリプトを使用して、アプリケーションで使用できる環境プロパティの値を確認できます。
> - 複数コンテナの Docker プラットフォームは、コンテナリソースの作成 AWS CloudFormation に を使用しません。そのため、このプラットフォームは AWS CloudFormation 関数を使用した環境プロパティの定義をサポートしていません。

Example .Ebextensions/sns-topic.config

```
Resources:
  NotificationTopic:
    Type: AWS::SNS::Topic

option_settings:
  aws:elasticbeanstalk:application:environment:
    NOTIFICATION_TOPIC: ``{"Ref" : "NotificationTopic"}``
```

また、この機能を使用して、AWS CloudFormation 擬似パラメータの情報を反映させることもできます。この例では、現在のリージョンを取得して、AWS_REGION という名前のプロパティに割り当てています。

Example .Ebextensions/env-regionname.config

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    AWS_REGION: ``{"Ref" : "AWS::Region"}``
```

ほとんどの Elastic Beanstalk プラットフォームでは、アプリケーションにリクエストを中継するリバースプロキシなど、インスタンス上で実行されるソフトウェア設定するオプションを指定して、その他の名前空間を定義します。プラットフォームで利用できる名前空間に関する詳細については、以下のいずれかのセクションを参照してください。

- Go – Go 設定の名前空間
- Java SE – Java SE 設定の名前空間
- Tomcat – Tomcat 設定の名前空間
- .NET Core on Linux – .NET Core on Linux 設定の名前空間
- .NET – aws:elasticbeanstalk:container:dotnet:apppool 名前空間
- Node.js – Node.js 設定の名前空間
- PHP – 設定の名前空間
- Python – Python 設定の名前空間
- Ruby – Ruby 設定の名前空間

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

環境プロパティへのアクセス

ほとんどの場合、アクセスするアプリケーションコードの環境プロパティは、環境変数に似ています。ただし、環境プロパティは、通常アプリケーションにのみ渡されるため、環境内のインスタンスに接続し、env を実行して表示することはできません。

- Go – `os.Getenv`

```
endpoint := os.Getenv("API_ENDPOINT")
```

- Java SE – `System.getenv`

```
String endpoint = System.getenv("API_ENDPOINT");
```

- Tomcat – `System.getProperty`および `System.getenv`

  2025 年 3 月 26 日以降にリリースされた Tomcat プラットフォームバージョンでは、`System.getenv`を使用してプレーンテキストの環境変数にアクセスすることもできます。引き続き `System.getProperty`を使用して、プレーンテキストの環境変数にアクセスできます。ただし、シークレットとして保存されている環境変数は、 を使用してのみ使用できます`System.getenv`。

```
String endpoint = System.getProperty("API_ENDPOINT");
```

```
String endpoint = System.getenv("API_ENDPOINT");
```

> **Important**
>
> 2025 年 3 月 26 日以降にリリースされた Tomcat プラットフォームバージョンの環境変数`System.getenv`へのアクセスを追加すると、Java システムプロパティよりも環境変数を優先するアプリケーションや、 から `System.getProperty`に明示的に切り替えるアプリケーションで予期しない動作が発生する可能性があります`System.getenv`。 https://docs.aws.amazon.com/elasticbeanstalk/latest/relnotes/release-2025-03-26-windows.html詳

細と推奨されるアクションについては、「Elastic Beanstalk Tomcat プラットフォームの使用」を参照してください。

- .NET Core on Linux – `Environment.GetEnvironmentVariable`

```
string endpoint = Environment.GetEnvironmentVariable("API_ENDPOINT");
```

- .NET – `appConfig`

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;
string endpoint = appConfig["API_ENDPOINT"];
```

- Node.js – `process.env`

```
var endpoint = process.env.API_ENDPOINT
```

- PHP – `$_SERVER`

```
$endpoint = $_SERVER['API_ENDPOINT'];
```

- Python – `os.environ`

```
import os
endpoint = os.environ['API_ENDPOINT']
```

- Ruby – `ENV`

```
endpoint = ENV['API_ENDPOINT']
```

デプロイ時に実行するスクリプトなど、アプリケーションコードを使用せずに環境プロパティにアクセスするには、`get-configプラットフォームスクリプト`を使用します。`get-config` を使用する設定の例については、GitHub の elastic-beanstalk-samples リポジトリを参照してください。

AWS X-Ray デバッグの設定

AWS Elastic Beanstalk コンソールまたは設定ファイルを使用して、環境内のインスタンスで AWS X-Ray デーモンを実行できます。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 AWS サービスであり、それを使用して、アプリケーションの問題と最適化の機会を特定するために使用できるサービスマップを構築します。

> **Note**
>
> 一部リージョンでは、X-Ray を使用できない場合があります。これらのいずれかのリージョンに環境を作成する場合は、環境内のインスタンスに対して X-Ray デーモンを実行することはできません。
> 各リージョンで提供される AWS サービスの詳細については、「リージョンテーブル」を参照してください。

X-Ray は、アプリケーションコードを設定するために使用できる SDK と、SDK から X-Ray API にデバッグ情報を中継するデーモンアプリケーションを提供します。

サポートされているプラットフォーム

X-Ray SDK は、次の Elastic Beanstalk プラットフォームで使用できます。

- Go - バージョン 2.9.1 以降
- Java 8 - バージョン 2.3.0 以降
- Java 8 と Tomcat 8 - バージョン 2.4.0 以降
- Node.js - バージョン 3.2.0 以降
- Windows Server - 2016 年 12 月 18 日以降にリリースされたすべてのプラットフォームのバージョン
- Python - バージョン 2.5.0 以降

サポートされるプラットフォームでは、設定オプションを使用して環境のインスタンスで X-Ray デーモンを実行できます。Elastic Beanstalk コンソールで、または設定ファイルを使用して、デーモンを有効にすることができます。

データを X-Ray アップロードするため、X-Ray デーモンは AWSXrayWriteOnlyAccess 管理ポリシーで IAM のアクセス許可を必要とします。これらのアクセス許可は Elastic Beanstalk インスタンスプロファイルに含まれています。デフォルトのインスタンスプロファイルを使用しない場合は、AWS X-Ray デベロッパーガイドの「X-Ray にデータを送信するためのアクセス許可をデーモンに付与する」を参照してください。

X-Ray を使用してデバッグするには、X-Ray SDK を使用する必要があります。手順とサンプルアプリケーションについては、AWS X-Ray デベロッパーガイドの「AWS X-Rayの開始方法」を参照してください。

デーモンを含まないプラットフォームのバージョンを使用している場合でも、設定ファイルでスクリプトを使ってデーモンを実行できます。詳細については、AWS X-Ray デベロッパーガイドの「X-Ray デーモンを手動でダウンロードして実行する (アドバンスト)」を参照してください。

セクション

- デバッグの設定
- aws:elasticbeanstalk: xray 名前空間

デバッグの設定

Elastic Beanstalk コンソールで実行中の環境で X-Ray デーモンを有効にできます。

Elastic Beanstalk コンソールでデバッグを有効にするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [Amazon X-Ray] セクションで、[アクティブ化] を選択します。
6. ページの最下部で [適用] を選択し変更を保存します。

環境の作成中にこのオプションを有効にすることもできます。詳細については、「新しい環境の作成ウィザード」を参照してください。

aws:elasticbeanstalk: xray 名前空間

`XRayEnabled` 名前空間の `aws:elasticbeanstalk:xray` オプションを使用してデバッグを有効にできます。

アプリケーションをデプロイする際に自動的にデバッグを有効にするには、ソースコードで設定ファイルのオプションを次のように設定します。

Example .ebextensions/debugging.config

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

Elastic Beanstalk 環境ログを表示する

AWS Elastic Beanstalk には、アプリケーションを実行する Amazon EC2 インスタンスのログを定期的に表示するための 2 つの方法があります。

- ローテーションされたインスタンスログを環境の Amazon S3 バケットにアップロードするように Elastic Beanstalk 環境を設定します。
- インスタンスログを Amazon CloudWatch Logs にストリーミングするように環境を設定します。

CloudWatch Logs へのインスタンスログストリーミングを設定する場合、Elastic Beanstalk は Amazon EC2 インスタンスでプロキシログとデプロイログ用の CloudWatch Logs ロググループを作成し、これらのログファイルをリアルタイムで CloudWatch Logs に転送します。インスタンスログの詳細については、「Elastic Beanstalk 環境の Amazon EC2 インスタンスからのログの表示」を参照してください。

インスタンスログに加えて、環境の拡張ヘルスを有効にすると、CloudWatch Logs にヘルス情報をストリーミングするように環境を設定できます。環境のヘルスステータスが変化すると、Elastic Beanstalk は新しいステータスと変更の原因の説明とともに、ヘルスロググループにレコードを追加します。環境ヘルスのストリーミングの詳細については、「Amazon CloudWatch Logs への Elastic Beanstalk 環境ヘルス情報のストリーミング」を参照してください。

インスタンスログ表示の設定

インスタンスログを表示するには、Elastic Beanstalk コンソールでインスタンスログのローテーションとログストリーミングを有効にします。

Elastic Beanstalk コンソールでインスタンスログのローテーションとログストリーミングを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。

4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [S3 ログストレージ] セクションで、[ログのローテーション] の下にある [アクティブ化] を選択して、ローテーションされたログを Amazon S3 にアップロードできるようにします。
6. [CloudWatch Logs へのインスタンスログのストリーミング] セクションで、以下を設定します。
   - [ログのストリーミング] – ログのストリーミングを有効にする場合は、[アクティブ化] を選択します。
   - [保持期間] – CloudWatch Logs でログを保持する日数を指定します。
   - [ライフサイクル] – 環境が終了している場合に、期限切れになるのを待たずに CloudWatch Logs から速やかにログを削除するには、[終了時にログを削除する] に設定します。
7. ページの最下部で [適用] を選択し変更を保存します。

ログストリーミングが有効になったら、[ソフトウェア] カテゴリまたはページに戻り、[ロググループ] のリンクを検索します。CloudWatch コンソールでインスタンスログを表示するには、このリンクをクリックします。

環境ヘルスログ表示の設定

環境ヘルスログを表示するには、Elastic Beanstalk コンソールで環境ヘルスログのストリーミングを有効にします。

Elastic Beanstalk コンソールで環境ヘルスログのストリーミングを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [モニタリング] セクションに移動します。
6. [CloudWatch Logs へのヘルスイベントストリーミング] で、以下を設定します。
   - [ログのストリーミング] – ログのストリーミングを有効にする場合は、[アクティブ化] を選択します。
   - [保持期間] – CloudWatch Logs でログを保持する日数を指定します。
   - [ライフサイクル] – 環境が終了している場合に、期限切れになるのを待たずに CloudWatch Logs から速やかにログを削除するには、[終了時にログを削除する] に設定します。
7. ページの最下部で [適用] を選択し変更を保存します。

ログ表示の名前空間

次の名前空間には、ログを表示の設定が含まれています。

- `aws:elasticbeanstalk:hostmanager` – ローテーションされたログを Amazon S3 にアップロードするよう設定します。
- `aws:elasticbeanstalk:cloudwatch:logs` – CloudWatch へのインスタンスログのストリーミングを設定します。
- `aws:elasticbeanstalk:cloudwatch:logs:health` CloudWatch への環境ヘルスのストリーミングを設定します。

Amazon SNS を使用した Elastic Beanstalk 環境の通知

Amazon Simple Notification Service (Amazon SNS) を使用して、アプリケーションに影響する重要なイベントを通知するように AWS Elastic Beanstalk 環境を設定できます。エラーが発生したり、環境の状態が変化したりする AWS たびに から E メールを受信するには、環境の作成時に E メールアドレスを指定します。

> **Note**
>
> Elastic Beanstalk は、通知に Amazon SNS を使用します。Amazon SNS の料金については、https://aws.amazon.com/sns/pricing/ を参照してください。

環境に関する通知を設定すると、それに関連する Amazon SNS トピックが、Elastic Beanstalk により自動的に作成されます。Amazon SNS トピックにメッセージを送信するには、Elastic Beanstalk が必要なアクセス許可を持っている必要があります。詳細については、「通知を送信するためのアクセス許可の設定」を参照してください。

重要なイベントが発生すると、Elastic Beanstalk はトピックにメッセージを送信します。そのメッセージは Amazon SNS に受信された後、トピックのサブスクライバーに中継されます。重要なイベントには、環境作成のエラーや環境およびインスタンスのヘルスステータスのあらゆる変化が含まれます。Amazon EC2 Auto Scaling のオペレーションに関するイベント (環境内のインスタンスの追加や削除など) や、その他の情報関連のイベントでは通知はトリガーされません。

ユーザーの E メールアドレスは、環境の作成時もしくはその後の任意の時点で、Elastic Beanstalk コンソールから入力することができます。これにより、Amazon SNS トピックが作成され、サブス

クライブされるようになります。Elastic Beanstalk ではトピックのライフサイクルが管理されており、環境が終了した場合や、環境マネジメントコンソールで E メールアドレスを削除した場合、トピックは削除されます。

`aws:elasticbeanstalk:sns:topics` 名前空間により、設定ファイルまたは CLI や SDK を使用して Amazon SNS トピックを設定するためのオプションが提供されます。これらの方法のいずれかを使用して、サブスクライバとエンドポイントのタイプを設定できます。サブスクライバーのタイプには、Amazon SQS キューまたは HTTP URL が選択可能です。

Amazon SNS 通知は、オン/オフの切り替えのみが可能です。トピックへの通知の送信頻度は、環境のサイズと構成に応じて高くなる場合があります。特定の状況に応じて通知を送信するように設定するための、他のオプションも用意されています。Amazon EventBridge でイベント駆動型ルールを設定しておくと、特定の基準を満たしたために Elastic Beanstalk がイベントを生成した場合の通知を受けられます。または、カスタムメトリクスを公開するように環境を設定し Amazon CloudWatch アラームを設定しておくと、メトリクスが異常なしきい値に達した場合に通知を出力させることも可能です。

Elastic Beanstalk コンソールを使用した通知の設定

Elastic Beanstalk コンソールにメールアドレスを入力することで、環境のための Amazon SNS トピックを作成できます。

Elastic Beanstalk コンソールで通知を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [E メール通知] セクションまで下にスクロールします。
6. メールアドレスを入力します。
7. ページの最下部で [適用] を選択し変更を保存します。

通知用の E メールアドレスを入力すると、Elastic Beanstalk は環境に関する Amazon SNS トピックを作成し、サブスクリプションを追加します。Amazon SNS は、サブスクライバ–のアドレスに E メールを送信し、サブスクリプションを確認します。サブスクリプションをアクティベーションして通知を受け取るには、確認メールに記載されたリンクをクリックする必要があります。

設定オプションを使用した通知の設定

環境に関する Amazon SNS 通知を設定するには、aws:elasticbeanstalk:sns:topics名前空間のオプションを使用します。これらのオプションは、設定ファイル、CLI、または SDK を使用して設定できます。

- 通知エンドポイント – 通知の送信先となる E メールアドレス、Amazon SQS キュー、または URL。このオプションを設定すると、指定されたエンドポイント用の SQS キューと指示書が作成されます。エンドポイントを E メールアドレスで指定しない場合は、`Notification Protocol` オプションも設定する必要があります。SNS は、`Notification Endpoint` の値に基づいて `Notification Protocol` の値を確認します。このオプションを複数回設定すると、トピックへの追加のサブスクリプションが作成されます。このオプションを削除した場合、トピックは削除されます。
- 通知プロトコル – `Notification Endpoint` に通知を送信するために使用されるプロトコル。このオプションのデフォルトは `email` です。このオプションを `email-json` に設定すると、JSON 形式の E メールを送信し、`http` または `https` に設定すると JSON 形式の通知を HTTP エンドポイントに投稿します。また、`sqs` に設定すると、SQS キューに通知を送信します。

  > **Note**
  >
  > AWS Lambda 通知はサポートされていません。

- 通知トピックの ARN – 環境に関する通知エンドポイントが設定された後、SNS トピックの ARN を取得するために読み取る情報。このオプションは、通知に関する既存の SNS トピックを使用するように設定することもできます。このオプションを使用して環境にアタッチしたトピックは、このオプションが変更されたり、環境が終了されても削除されません。

  Amazon SNS 通知を設定するには、適切ななアクセス許可が必要です。IAM ユーザーが Elastic Beanstalk の管理ユーザーポリシー AdministratorAccess-AWSElasticBeanstalk を使用する場合には、その環境向けに Elastic Beanstalk によって作成されるデフォルトの Amazon SNS トピックを設定するためのアクセス許可が既に付与されています。ただし、Elastic Beanstalk で管理されない Amazon SNS トピックを設定する場合は、次のポリシーをご使用のユーザーロールに追加する必要があります。

  JSON

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

```
    {
      "Effect": "Allow",
      "Action": [
        "sns:SetTopicAttributes",
        "sns:GetTopicAttributes",
        "sns:Subscribe",
        "sns:Unsubscribe",
        "sns:Publish"
      ],
      "Resource": [
        "arn:aws:sns:us-east-2:123456789012:sns_topic_name"
      ]
    }
  ]
}
```

- 通知トピック名 – 環境の通知に使用される Amazon SNS トピックの名前を、カスタマイズする際に使用されるオプション。同じ名前のトピックが既に存在する場合、Elastic Beanstalk はそのトピックを環境にアタッチします。

> **Warning**
>
> `Notification Topic Name` を使用して既存の SNSトピックを環境にアタッチすると、以後に環境を終了したりこの設定を変更したタイミングで、Elastic Beanstalk は当該トピックを削除します。

このオプションを変更すると `Notification Topic ARN` も変更されます。トピックが既に環境にアタッチされている場合、古いトピックは Elastic Beanstalk により削除され、新しいトピックとサブスクリプションが作成されます。

外部で作成されたカスタムトピックを使用する場合は、そのカスタムトピックの名前に加えて ARN も指定する必要があります。管理ユーザーポリシーでは、カスタム名を持つトピックは自動的に検出されません。つまり、IAM ユーザーにはそのための Amazon SNS アクセス許可を付与する必要があります。使用するポリシーはカスタムトピック ARN にされるものと類似していますが、次の項目を追加します。

- `Actions`リストにはさらに 2 つのアクション (具体的には `sns:CreateTopic` および `sns:DeleteTopic`) を含めます。
- `Notification Topic Name` を別のカスタムトピック名に変更する場合は、それら両方のトピックの ARN を `Resource` リストに含める必要があります。または、両方をカバーする正規

表現を含めることもできます。このように Elastic Beanstalk には、古いトピックを削除して新しいトピックを作成するための権限が付与されています。

EB CLI および Elastic Beanstalk コンソールでは、上記のオプションに推奨値が適用されます。設定ファイルを使用して同じファイルを設定する場合は、これらの設定を削除する必要があります。詳細については、「推奨値」を参照してください。

通知を送信するためのアクセス許可の設定

このセクションでは、Amazon SNS を使用する通知に関連するセキュリティ上の考慮事項について説明します。これには、次のように 2 つの異なるケースがあります。

- Elastic Beanstalk により環境用に作成された、デフォルトの Amazon SNS トピックを使用する場合。
- 設定オプションを使用して、外部から Amazon SNS トピックを導入する場合。

Amazon SNS トピックのデフォルトのアクセスポリシーでは、トピック所有者のみがトピックを公開またはサブスクライブできます。ただし、ポリシーを適切に設定することで、このセクションで説明されている 2 つのケースのいずれかにおいて、Elastic Beanstalk に対し Amazon SNS トピックへの公開を許可することが可能になります。この後のサブセクションで、その詳細について説明します。

デフォルトトピックのアクセス許可

環境に関する通知を設定すると、Elastic Beanstalk は環境の Amazon SNS トピックを作成します。Amazon SNS トピックにメッセージを送信するには、Elastic Beanstalk が必要なアクセス許可を持っている必要があります。環境内で、Elastic Beanstalk コンソールまたは EB CLI によって生成されたサービスロールを使用しているか、アカウントのモニタリングサービスにリンクされたロールを使用している場合には、他に何もする必要はありません。これらの管理ロールには、Elastic Beanstalk が Amazon SNS トピックにメッセージを送信するために必要なアクセス許可が含まれています。

ただし、環境の作成時にカスタムサービスロールを設定した場合は、そのカスタムサービスロールに以下のポリシーが含まれていている必要があります。

JSON

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": [
        "arn:aws:sns:us-east-2:123456789012:ElasticBeanstalkNotifications*"
      ]
    }
  ]
}
```

外部トピックのアクセス許可

「設定オプションを使用した通知の設定」では、Elastic Beanstalk により提供された Amazon SNS トピックを、別の Amazon SNS トピックに置き換える方法について説明しています。SNS トピックを置き換えた場合、そのトピックに対する公開がユーザーに許可されているかが、Elastic Beanstalk により確認されます。このアクセス許可は、そのトピックを環境に関連付けるために必要となります。ユーザーには `sns:Publish` が必要です。サービスロールでも同じアクセス許可を使用します。この状態となっているか確認するため、Elastic Beanstalk は環境を作成または更新するアクションの中で、テスト通知を SNS に送信します。このテストが失敗すると、当該の環境の作成または更新も失敗します。Elastic Beanstalk からは、この失敗の原因を報告するメッセージが表示されます。

環境にカスタムサービスロールを導入する場合は、そのカスタムサービスロールに次のポリシーが含まれていることを確認してください。これにより、Elastic Beanstalk が Amazon SNS トピックにメッセージを送信できるようになります。次に示すコードでは、*`sns_topic_name`* を、設定オプションで指定した Amazon SNS トピックの名前に置き換えています。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
```

```
      "Resource": [
        "arn:aws:sns:us-east-2:123456789012:sns_topic_name"
      ]
    }
  ]
}
```

Amazon SNS アクセスコントロールの詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS アクセスコントロールのケース例」を参照してください。

Elastic Beanstalk を使用した Amazon Virtual Private Cloud (Amazon VPC) の設定

Amazon Virtual Private Cloud (Amazon VPC) は、Elastic Beanstalk でアプリケーションを実行する EC2 インスタンスにトラフィックを安全にルーティングするネットワークサービスです。環境起動時に VPC を設定しない場合、Elastic Beanstalk はデフォルトの VPC を使用します。

カスタム VPC で環境を起動して、ネットワーキングおよびセキュリティの設定をカスタマイズできます。Elastic Beanstalk では、リソースに使用するサブネットを選択でき、環境で使用するインスタンスおよびロードバランサーの IP アドレスを設定する方法を提供します。作成時には環境は VPC にロックされていますが、実行中の環境でサブネットおよび IP アドレスの設定を変更できます。

Elastic Beanstalk コンソールでの VPC 設定の定義

環境作成時にカスタム VPC を選択した場合は、Elastic Beanstalk コンソールでその VPC 設定を変更できます。

環境の VPC 設定を構成するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [ネットワーク] 設定カテゴリで、[編集] を選択します。

以下の設定を使用できます。

オプション

- VPC
- ロードバランサーの可視性
- ロードバランサーのサブネット
- インスタンスのパブリック IP アドレス
- インスタンスのサブネット
- データベースのサブネット

VPC

環境の VPC を選択します。環境作成時のみこの設定を変更することができます。

ロードバランサーの可視性

負荷分散された環境で、ロードバランサーのスキームを選択します。デフォルトでは、ロードバランサーはパブリック IP アドレスおよびドメイン名があるパブリックです。アプリケーションが VPC 内あるいは接続した VPN からのトラフィックのみを配信する場合には、このオプションの選択を解除し、ロードバランサーにプライベートサブネットを選択してロードバランサーを内部向けにし、インターネットからのアクセスを無効にします。

ロードバランサーのサブネット

負荷分散された環境で、ロードバランサーがトラフィックを配信するために使用するサブネットを選択します。パブリックアプリケーションで、パブリックサブネットを選択します。マルチアベイラビリティーゾーンにサブネットを使用して、高可用性に対応します。内部アプリケーションの場合は、プライベートサブネットを選択し、ロードバランサーの可視性を無効にします。

インスタンスのパブリック IP アドレス

アプリケーションインスタンスにパブリックサブネットを選択する場合、パブリック IP アドレスを有効にしてインターネットからルーティングできるようにします。

インスタンスのサブネット

アプリケーションインスタンスのサブネットを選択します。ロードバランサーが使用する各アベイラビリティーゾーンに少なくとも 1 つのサブネットを選択します。インスタンスにプライベートサブ

ネットを選択した場合は、VPC にインターネットにアクセスするためにインスタンスが使用できるパブリックサブネットでの NAT ゲートウェイがあることが必要です。

データベースのサブネット

Elastic Beanstalk 環境にアタッチされた Amazon RDS データベースを実行する場合は、データベースインスタンスのサブネットを選択します。高可用性を実現するために、データベースのマルチ AZ を作成し、各アベイラビリティーゾーンにサブネットを選択します。アプリケーションがデータベースに接続できるようにするには、同じサブネットで両方を実行します。

aws:ec2:vpc 名前空間

`aws:ec2:vpc` 名前空間の設定オプションを使用して、環境のネットワーク設定を構成します。

以下の設定ファイルは、この名前空間のオプションを使用してパブリック/プライベート設定で環境の VPC およびサブネットを設定します。設定ファイルで VPC ID を設定するには、環境作成時にこのファイルがアプリケーションソースバンドルに含まれている必要があります。環境作成時にこの設定を定義するその他の方法については、「環境の作成時の設定オプションの設定」を参照してください。

Example .ebextensions/vpc.config - Public-private

```
option_settings:
   aws:ec2:vpc:
      VPCId: vpc-087a68c03b9c50c84
      AssociatePublicIpAddress: 'false'
      ELBScheme: public
      ELBSubnets: subnet-0fe6b36bcb0ffc462,subnet-032fe3068297ac5b2
      Subnets: subnet-026c6117b178a9c45,subnet-0839e902f656e8bd1
```

この例では、ロードバランサーおよび EC2 インスタンスが同じパブリックサブネットを実行している場合のパブリック/パブリック設定を示しています。

Example .ebextensions/vpc.config - Public-public

```
option_settings:
   aws:ec2:vpc:
      VPCId: vpc-087a68c03b9c50c84
      AssociatePublicIpAddress: 'true'
      ELBScheme: public
```

```
    ELBSubnets: subnet-0fe6b36bcb0ffc462,subnet-032fe3068297ac5b2
    Subnets: subnet-0fe6b36bcb0ffc462,subnet-032fe3068297ac5b2
```

EC2-Classic から VPC への Elastic Beanstalk 環境の移行

このトピックでは、Elastic Beanstalk 環境を EC2-Classic ネットワークプラットフォームから Amazon Virtual Private Cloud (Amazon VPC) ネットワークに移行する方法のさまざまなオプションについて説明します。

2013 年 12 月 4 日より前に AWS アカウントを作成した場合は、一部の で EC2-Classic ネットワーク設定を使用する環境がある場合があります AWS リージョン。2013 年 12 月 4 日以降に作成されたすべての AWS アカウントは、すべての AWS リージョンで既に VPC 専用です。例外は、サポートリクエストの結果として Amazon EC2-Classic が有効になっている場合のみです。

> **Note**
>
> 環境のネットワーク設定は、Elastic Beanstalk コンソールの [設定概要] ページの [Network configuration] (ネットワーク設定) カテゴリで表示できます。

移行が必要な理由

Amazon EC2-Classic は 2022 年 8 月 15 日に標準サポートを終了する予定です。ワークロードの中断を避けるために、2022 年 8 月 15 日までに Amazon EC2-Classic から VPC に移行することをお勧めします。また、今後 Amazon EC2-Classic で AWS リソースを起動せず、代わりに Amazon VPC を使用することをリクエストします。

Elastic Beanstalk 環境を Amazon EC2-Classic から Amazon VPC に移行する場合は、新しい AWS アカウントを作成する必要があります。また、 AWS 新しい AWS アカウントで EC2-Classic 環境を再作成する必要があります。デフォルトの VPC を使用するために、環境に追加の設定作業を行う必要はありません。デフォルトの VPC が要件を満たさない場合は、カスタム VPC を手動で作成し、環境に関連付けします。

または、既存の AWS アカウントに新しい AWS アカウントに移行できないリソースがある場合は、現在のアカウントに VPC を追加します。次に、VPC を使用するように環境を設定します。

詳細については、ブログ記事「EC2-Classic Networking は販売終了になります — 準備方法はこちら」を参照してください。

EC2-Classic から新しい AWS アカウントに環境を移行する (推奨)

2013 年 12 月 4 日以降に作成された AWS アカウントがまだない場合は、新しいアカウントを作成します。自分の環境をこの新しいアカウントに移行します。

1. 新しいアカウントは、その環境にデフォルトの VPC AWS を提供します。カスタム VPC を作成する必要がない場合は、ステップ 2 に進みます。

   カスタム VPC は、次のいずれかの方法で作成できます。

   - 使用可能な設定オプションの 1 つを使用して、VPC コンソールウィザードで VPC をすばやく作成する。詳細については、「Amazon VPC コンソールウィザードの設定」を参照してください。
   - VPC に対してより具体的な要件がある場合、Amazon VPC コンソールでカスタム VPC を作成する。例えば、ユースケースで特定の数のサブネットが必要な場合など、これを行うことをお勧めします。詳細については、「VPC とサブネット」を参照してください。
   - Elastic Beanstalk 環境で テンプレートを使用する場合は、GitHub ウェブサイトの elastic-beanstalk-samples リポジトリを使用して VPC を作成します。 GitHub AWS CloudFormation このリポジトリには AWS CloudFormation テンプレートが含まれています。詳細については、「Amazon VPC で Elastic Beanstalk を使用する」を参照してください。

   > **Note**
   >
   > 新しい環境の作成ウィザードを使用して、新しい AWS アカウントで環境を再作成するときに、カスタム VPC を作成することもできます。ウィザードを使用してカスタム VPC を作成しようとすると、ウィザードによって Amazon VPC コンソールにリダイレクトされます。

2. 新しい AWS アカウントで、新しい環境を作成します。環境には、移行元の AWS アカウントの既存の環境と同じ設定を含めることをお勧めします。これを行うには、次のいずれかのアプローチを使用します。

   > **Note**
   >
   > 移行後も新しい環境で同じ CNAME を使用する必要がある場合は、EC2-Classic プラットフォームの元の環境を終了します。これにより、CNAME がリリースされ、使用できるようになります。ただし、そうすると、その環境のダウンタイムが発生し、EC2-Classic 環境の終了から新しい環境の作成までの間に、別のお客様がその CNAME を選

択する可能性があるというリスクが生じます。詳細については、「Elastic Beanstalk 環境を終了する」を参照してください。
独自のドメイン名を持つ環境では、CNAME にこの問題はありません。ドメインネームシステム (DNS) を更新するだけで、リクエストを新しい CNAME に転送できます。

- Elastic Beanstalk コンソールの新しい環境の作成ウィザードを使用します。ウィザードには、カスタム VPC を作成するオプションがあります。カスタム VPC を作成しない場合は、デフォルトの VPC が割り当てられます。
- Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用して、新しい AWS アカウントに環境を再作成します。eb create コマンドの説明の例の 1 つは、カスタム VPC での環境の作成を示しています。VPC ID を指定しない場合、環境ではデフォルトの VPC が使用されます。

  この方法を使用すると、2 つの AWS アカウント間で保存された設定ファイルを使用できます。このため、すべての設定情報を手動で入力する必要はありません。ただし、eb config save コマンドを使用して、移行する EC2-Classic 環境の設定を保存する必要があります。保存した設定ファイルを、新しいアカウントの環境の新しいディレクトリにコピーします。

  > **Note**
  > 新しいアカウントで使用する前に、保存した設定ファイルのデータの一部を編集する必要があります。また、以前のアカウントに関連する情報を、新しいアカウントの正しいデータで更新する必要があります。例えば、 AWS Identity and Access Management (IAM) ロールの Amazon リソースネーム (ARN) を新しいアカウントの IAM ロールの ARN に置き換える必要があります。

  `cfg` を指定して eb create コマンドを実行すると、指定した保存済み設定ファイルを使用して新しい環境が作成されます。詳細については、「Elastic Beanstalk の保存された設定を使用する」を参照してください。

同じ AWS アカウント内の EC2-Classic から環境を移行する

既存のアカウントには、新しい AWS アカウントに移行できないリソースがある AWS 場合があります。この場合、環境を再作成し、作成する環境ごとに VPC を手動で設定する必要があります。

環境をカスタム VPC に移行する

前提条件

開始する前に、VPC が必要です。デフォルト以外の (カスタム) VPC は、次のいずれかの方法で作成できます。

- 使用可能な設定オプションの 1 つを使用して、VPC コンソールウィザードで VPC をすばやく作成する。詳細については、「Amazon VPC コンソールウィザードの設定」を参照してください。
- VPC に対してより具体的な要件がある場合、Amazon VPC コンソールでカスタム VPC を作成する。例えば、ユースケースで特定の数のサブネットが必要な場合など、これを行うことをお勧めします。詳細については、「VPC とサブネット」を参照してください。
- Elastic Beanstalk 環境で テンプレートを使用する場合は、GitHub ウェブサイトの elastic-beanstalk-samples リポジトリを使用して VPC を作成します。 GitHub AWS CloudFormation このリポジトリには AWS CloudFormation テンプレートが含まれています。詳細については、「Amazon VPC で Elastic Beanstalk を使用する」を参照してください。

次のステップでは、新しい環境で VPC を設定するときに、生成された VPC ID とサブネット ID を使用します。

1. 既存の環境と同じ設定を含む新しい環境を作成します。これを行うには、次のいずれかのアプローチを使用します。

   > **Note**
   >
   > 保存された設定の機能は、新しいアカウントで環境を再作成するのに役立ちます。この機能は環境の設定を保存できるため、他の環境を作成または更新するときに適用できます。詳細については、「Elastic Beanstalk の保存された設定を使用する」を参照してください。

   - Elastic Beanstalk コンソールを使用して、新しい環境を設定するときに、EC2-Classic 環境から保存された設定を適用します。この設定では VPC が使用されます。詳細については、「Elastic Beanstalk の保存された設定を使用する」を参照してください。
   - Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用し、eb create コマンドを実行して環境を再作成します。元の環境のパラメータと VPC 識別子を指定します。eb

create コマンドの説明の例の 1 つは、カスタム VPC で環境を作成する方法を示しています。

- AWS Command Line Interface（AWS CLI) を使用し、 elasticbeanstalk create-environment コマンドを使用して環境を再作成します。元の環境のパラメータと VPC 識別子を指定します。手順については、AWS CLI を使用した Elastic Beanstalk 環境の作成 を参照してください。

2. 現在の環境の CNAME を新しい環境と入れ替えます。これにより、作成した新しい環境を、使い慣れたアドレスで参照できるようになります。EB CLI または AWS CLIを使用できます。
   - EB CLI を使用する場合、eb swap コマンドを実行して、環境 CNAME を入れ替えます。詳細については、「Elastic Beanstalk を管理するための EB コマンドラインインターフェイス (EB CLI) の設定」を参照してください。
   - を使用して AWS CLI、環境 CNAMEs を elasticbeanstalk swap-environment-cnames コマンドに置き換えます。詳細については、『AWS CLI コマンドリファレンス』を参照してください。

Elastic Beanstalk 環境のドメイン名

デフォルトでは、お使いの環境は `elasticbeanstalk.com` のサブドメインのユーザーが使用できます。環境を作成する際に、アプリケーションのホスト名を選択できます。ドメインとサブドメインは *`region`*`.elasticbeanstalk.com` に自動入力されます。

ユーザーを環境にルーティングするために、Elastic Beanstalk はお使いの環境のロードバランサーを指す CNAME レコードを登録します。Elastic Beanstalk コンソールの環境の概要ページで、CNAME の現在の値を含む環境のアプリケーションの URL を確認できます。

概要ページで URL を選択するか、ナビゲーションペインで [Go to environment (環境へ移動)] を選択して、アプリケーションのウェブページに移動します。

別の環境の CNAME と交換することで、お使いの環境の CNAME を変更できます。手順については、「Elastic Beanstalk を使用したブルー/グリーンデプロイ」を参照してください。

ドメイン名を所有している場合は、Amazon Route 53 を使用してお使いの環境で解決できます。Amazon Route 53 でドメイン名を購入することも、別のプロバイダから購入したものを使用することもできます。

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。

カスタムドメインの使用の詳細については、Amazon Route 53 デベロッパーガイドの「AWS Elastic Beanstalk 環境 へのトラフィックのルーティング」を参照してください。

> **Important**
>
> 環境を終了する場合は、作成した CNAME マッピングも削除する必要があります。これにより、使用可能になったホスト名を他のお客様が再利用できます。DNS エントリのダングリングを防ぐため、終了した環境を指す DNS レコードを必ず削除してください。DNS エントリがダングリングしていると、ユーザーのドメイン宛のインターネットトラフィックがセキュリティの脆弱性にさらされる可能性があります。また、他のリスクをもたらす可能性もあります。
>
> 詳細については、Amazon Route 53 デベロッパーガイドの「Route 53 でのダングリング委任レコードからの保護」を参照してください。また、ダングリング DNS エントリの詳細については、AWS セキュリティブログの 「Enhanced Domain Protections for Amazon CloudFront Requests」を参照してください。

Elastic Beanstalk 環境の設定 (高度)

AWS Elastic Beanstalk 環境を作成すると、Elastic Beanstalk はアプリケーションの実行とサポートに必要なすべての AWS リソースをプロビジョニングして設定します。環境のメタデータの設定や動作の更新に加えて、設定オプションの値を指定することによって、これらのリソースをカスタマイズできます。例えば、Amazon SQS キューおよびキューの深さに対するアラームを追加したり、Amazon ElastiCache クラスターを追加したりする場合です。

設定オプションのほとんどで、Elastic Beanstalk に自動的に適用されるデフォルト値が設定されています。これらのデフォルト設定は、設定ファイル、保存済み設定、コマンドラインオプションで、または Elastic Beanstalk API を直接呼び出して上書きできます。EB CLI および Elastic Beanstalk コンソールでは、一部のオプションに推奨値も適用されます。

ソースバンドルと共に設定ファイルを含めることにより、アプリケーションバージョンのデプロイと同時に環境を簡単にカスタマイズできます。インスタンスのソフトウェアをカスタマイズするときは、カスタム AMI を作成するより設定ファイルを使用する方が、AMI のセットを保持する必要がないので優っています。

アプリケーションをデプロイするときは、アプリケーションが依存するソフトウェアをカスタマイズおよび設定したい場合があります。このようなファイルとしては、アプリケーションで必要な依存関係（yum リポジトリからの追加パッケージなど）や、 AWS Elastic Beanstalkでデフォルトの特定の設定を上書きするための httpd.conf の代わりの設定ファイルなどがあります。

トピック

- 設定オプション
- 設定ファイル (.ebextensions) による高度な環境のカスタマイズ
- Elastic Beanstalk の保存された設定を使用する
- マニフェスト環境 (env.yaml)
- Elastic Beanstalk 環境でのカスタム Amazon マシンイメージ (AMI) の使用
- 静的ファイルの提供
- Elastic Beanstalk 環境の HTTPS の設定

設定オプション

Elastic Beanstalk は、環境の動作、およびその環境に含まれるリソースの設定に使用する多数の設定オプションを定義します。設定オプションは、`aws:autoscaling:asg` のような名前空間に構成されており、環境の Auto Scaling グループのオプションを定義します。

環境を作成する際、Elastic Beanstalk コンソールおよび EB CLI で、明示的に設定するオプションや、クライアントによって定義されている推奨値などの設定オプションを設定します。また、保存済み設定や、設定ファイル内の設定オプションを設定することもできます。複数の場所に同一のオプションが設定されている場合、使用する値は優先順位によって判断されます。

設定オプションの設定は、テキスト形式で構成し、環境の作成前に保存し、サポートされる任意のクライアントを使用した環境の作成時に適用し、環境の作成後に追加、変更または削除できます。これら 3 つのステージでの設定オプションの操作で使用可能なすべての方法の詳細については、次のトピックを参照してください。

- 環境を作成する前に設定オプションを設定する
- 環境の作成時の設定オプションの設定
- 環境の作成後に設定オプションを設定する

各項目のデフォルト値やサポートされる値を含む名前空間とオプションの完全なリストについては、「すべての環境に対する汎用オプション」と「プラットフォーム固有のオプション」を参照してください。

優先順位

環境の作成時には、複数のソースの設定オプションが、次のように優先順位の高いものから適用されます。

- 環境に直接適用される設定 – Elastic Beanstalk コンソール、EB CLI、 AWS CLI SDKs など、任意のクライアントによって Elastic Beanstalk API で環境の作成または環境の更新オペレーション中に指定された設定。Elastic Beanstalk コンソールおよび EB CLI は、一部のオプションで推奨値も適用します。推奨値は、上書きされるまでこのレベルで適用されます。
- 保存済み設定 – 指定されている場合、環境に直接適用されないオプションの設定が、保存済み設定からロードされます。

- 設定ファイル (.ebextensions) – 環境に直接適用されず、保存済み設定でも指定されていないオプションの設定が、アプリケーションソースバンドルのルートにある `.ebextensions` フォルダの設定ファイルからロードされます。

  設定ファイルはアルファベット順に実行されます。たとえば、`.ebextensions/01run.config` は `.ebextensions/02do.config` の前に実行します。
- デフォルト値 – 設定オプションにデフォルト値がある場合、オプションが上記のいずれのレベルにも設定されていない場合にのみ適用されます。

1 つ以上の場所で同じ設定オプションが定義されている場合は、最も優先順位の高いものが適用されます。保存済み設定から設定が適用されるか、設定が直接環境に適用されると、設定は環境の設定の一部として保存されます。これらの設定は、AWS CLI や EB CLI を使用して削除できます。

設定ファイルの設定は、環境に直接適用されず、設定ファイルの変更と新しいアプリケーションバージョンのデプロイなしには削除できません。その他の方法の 1 つで適用された設定が削除されると、ソースバンドルの設定ファイルから同じ設定がロードされます。

例えば、環境の作成時に Elastic Beanstalk コンソール、コマンドラインオプション、保存済み設定のいずれかを使用して、環境内のインスタンスの最小数を 5 に設定したとします。アプリケーションのソースバンドルには、インスタンスの最小数を 2 に設定する設定ファイルも含まれます。

環境を作成すると、Elastic Beanstalk は `MinSize` 名前空間内の `aws:autoscaling:asg` オプションを 5 に設定します。その後、環境設定からオプションを削除すると、設定ファイルの値がロードされ、インスタンスの最小数は [2] に設定されます。その後、ソースバンドルから設定ファイルを削除して再度デプロイすると、Elastic Beanstalk はデフォルト設定の [1] を使用します。

推奨値

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) および Elastic Beanstalk コンソールでは、一部の設定オプションの推奨値が表示されます。これらの値はデフォルト値と異なる場合があり、環境の作成時に API レベルで設定されます。推奨値により、Elastic Beanstalk は、API への後方互換性のない変更なしに、デフォルトの環境設定を向上させることができます。

例えば、EB CLI と Elastic Beanstalk コンソールの両方が、EC2 インスタンスタイプの設定オプションを設定したとします (名前空間 `InstanceType` の `aws:autoscaling:launchconfiguration`)。各クライアントは、固有の方法でデフォルト設定を上書きします。コンソールでは、[Create New Environment] ウィザードの [Configuration Details] ページで、ドロップダウンメニューからさまざまなインスタンスタイプを選択できます。EB CLI では、`--instance_type` の eb create パラメータを使用できます。

推奨値は API レベルで設定されているため、設定ファイルや保存済み設定で設定された同じオプションの値を上書きします。以下のオプションが設定されます。

Elastic Beanstalk コンソール

- 名前空間: `aws:autoscaling:launchconfiguration`

  オプション名:`IamInstanceProfile` 、`EC2KeyName`、`InstanceType`
- 名前空間: `aws:autoscaling:updatepolicy:rollingupdate`

  オプション名: `RollingUpdateType`、`RollingUpdateEnabled`
- 名前空間: `aws:elasticbeanstalk:application`

  オプション名: `Application Healthcheck URL`
- 名前空間: `aws:elasticbeanstalk:command`

  オプション名: `DeploymentPolicy` および `BatchSize`、`BatchSizeType`
- 名前空間: `aws:elasticbeanstalk:environment`

  オプション名: `ServiceRole`
- 名前空間: `aws:elasticbeanstalk:healthreporting:system`

  オプション名: `SystemType`、`HealthCheckSuccessThreshold`
- 名前空間: `aws:elasticbeanstalk:sns:topics`

  オプション名: `Notification Endpoint`
- 名前空間: `aws:elasticbeanstalk:sqsd`

  オプション名: `HttpConnections`
- 名前空間: `aws:elb:loadbalancer`

  オプション名: `CrossZone`
- 名前空間: `aws:elb:policies`

  オプション名: `ConnectionDrainingTimeout`、`ConnectionDrainingEnabled`

EB CLI

- 名前空間: `aws:autoscaling:launchconfiguration`

オプション名: `IamInstanceProfile`、`InstanceType`

- 名前空間: `aws:autoscaling:updatepolicy:rollingupdate`

  オプション名: `RollingUpdateType`、`RollingUpdateEnabled`

- 名前空間: `aws:elasticbeanstalk:command`

  オプション名: `BatchSize`、`BatchSizeType`

- 名前空間: `aws:elasticbeanstalk:environment`

  オプション名: `ServiceRole`

- 名前空間: `aws:elasticbeanstalk:healthreporting:system`

  オプション名: `SystemType`

- 名前空間: `aws:elb:loadbalancer`

  オプション名: `CrossZone`

- 名前空間: `aws:elb:policies`

  オプション名: `ConnectionDrainingEnabled`

環境を作成する前に設定オプションを設定する

AWS Elastic Beanstalk は、環境内のリソースに適用される設定を変更できる多数の設定オプションをサポートしています。これらのオプションの一部には、環境のカスタマイズで上書きされるデフォルト値があります。その他のオプションを設定すると、追加機能を有効化できます。

Elastic Beanstalk では、設定オプションの設定を保存する 2 つの方法をサポートしています。YAML または JSON 形式の設定ファイルは、`.ebextensions` というディレクトリ名でアプリケーションのソースコードに含め、アプリケーションソースバンドルの一部としてデプロイできます。設定ファイルをローカルで作成して管理します。

保存済み設定とは、実行中の環境や JSON オプション ファイルから作成され、Elastic Beanstalk に保存されるテンプレートです。既存の保存済み設定を拡張し、新しい設定を作成することもできます。

> **Note**
>
> 設定ファイルや保存済み設定で定義された設定は、環境の作成時または環境の作成後に設定された設定（Elastic Beanstalk コンソールや EB CLI に適用される推奨値など）よりも優先順位が低くなります。詳細については、「優先順位」を参照してください。

オプションは JSON ドキュメントでも指定され、EB CLI や で環境を作成または更新する際に Elastic Beanstalk に対して直接設定される場合があります AWS CLIこの方法で Elastic Beanstalk に直接設定されるオプションは、他の方法によるすべてのオプションを上書きします。

使用可能なオプションの詳細なリストについては、 を参照してください設定オプション

メソッド

- 設定ファイル (.ebextensions)
- 保存された設定
- JSON ドキュメント
- EB CLI 設定

設定ファイル (`.ebextensions`)

アプリケーションを機能させるのに必要なオプションを `.ebextensions` を使用して設定し、優先順位の高い設定に上書きされるその他のオプションのデフォルト値を指定します。`.ebextensions` で指定されるオプションは、優先順位が一番低く、他のあらゆるレベルの設定に上書きされます。

設定ファイルを使用するには、プロジェクトのソースコードの最上位に `.ebextensions` という名前のフォルダを作成します。拡張子 `.config` をつけてファイルを追加し、次の方法でオプションを指定します。

```
option_settings:
  - namespace:  namespace
    option_name:  option name
    value:  option value
  - namespace:  namespace
    option_name:  option name
    value:  option value
```

たとえば次の設定ファイルは、アプリケーションのヘルスチェック URL を /health に設定します。

healthcheckurl.config

```
option_settings:
  - namespace:  aws:elasticbeanstalk:application
    option_name:  Application Healthcheck URL
    value:  /health
```

JSON の場合:

```
{
 "option_settings" :
    [
      {
        "namespace" : "aws:elasticbeanstalk:application",
        "option_name" : "Application Healthcheck URL",
        "value" : "/health"
      }
    ]
}
```

これにより、Elastic Beanstalk 環境の Elastic Load Balancing ロードバランサーが、各 EC2 インスタンスへのパス /health への HTTP リクエストを行い、それが正常であるかどうかを判断するように設定します。

> **Note**
>
> YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

.ebextensions ディレクトリをアプリケーションソースバンドルに含め、新規または既存の Elastic Beanstalk 環境にデプロイします。

設定ファイルは、環境内のサーバーで実行されるソフトウェアやファイルのカスタマイズのために、option_settings に加えて複数のセクションをサポートします。詳細については、「」を参照してください.Ebextensions

保存された設定

保存済み設定を作成し、Elastic Beanstalk コンソール、EB CLI、または による環境の作成時または作成後に既存の環境に適用した設定を保存します AWS CLI保存済み設定はアプリケーションに属し、同アプリケーションの新しいまたは既存の環境に適用される場合があります。

クライアント

- Elastic Beanstalk コンソール
- EB CLI
- AWS CLI

Elastic Beanstalk コンソール

保存した設定を作成するには (Elastic Beanstalk コンソール)、

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Save Configuration] (設定の保存) の順に選択します。
4. 画面上のダイアログボックスを使用して、アクションを完了します。

保存された設定は、アプリケーションに関連する名前のフォルダ内の Elastic Beanstalk S3 バケットに格納されます。例えば、us-west-2 リージョンにあるアカウント番号 123456789012 の `my-app` という名前のアプリケーションの設定は、`s3://elasticbeanstalk-us-west-2-123456789012/resources/templates/my-app` にあります。

EB CLI

EB CLI は、eb config で保存済み設定とやりとりするためのサブコマンドも指定します。

保存済み設定を作成する (EB CLI)

1. アタッチされた環境の現在の設定を保存する

   ```
   ~/project$ eb config save --cfg my-app-v1
   ```

   EB CLI は、設定を `~/project/.elasticbeanstalk/saved_configs/my-app-v1.cfg.yml` に保存します。

2. 保存済み設定を必要に応じてローカルで変更します。
3. 保存済み設定を S3 にアップロードします。

```
~/project$ eb config put my-app-v1
```

AWS CLI

実行中の環境から、`aws elasticbeanstalk create-configuration-template` で保存済み設定を作成します。

保存済み設定を作成するには (AWS CLI)

1. Elastic Beanstalk 環境の環境IDを `describe-environments` で特定します。

```
$ aws elasticbeanstalk describe-environments --environment-name my-env
{
    "Environments": [
        {
            "ApplicationName": "my-env",
            "EnvironmentName": "my-env",
            "VersionLabel": "89df",
            "Status": "Ready",
            "Description": "Environment created from the EB CLI using \"eb create
\"",
            "EnvironmentId": "e-vcghmm2zwk",
            "EndpointURL": "awseb-e-v-AWSEBLoa-1JUM8159RA11M-43V6ZI1194.us-
west-2.elb.amazonaws.com",
            "SolutionStackName": "64bit Amazon Linux 2015.03 v2.0.2 running Multi-
container Docker 1.7.1 (Generic)",
            "CNAME": "my-env-nfptuqaper.elasticbeanstalk.com",
            "Health": "Green",
            "AbortableOperationInProgress": false,
            "Tier": {
                "Version": " ",
                "Type": "Standard",
                "Name": "WebServer"
            },
            "HealthStatus": "Ok",
            "DateUpdated": "2015-10-01T00:24:04.045Z",
            "DateCreated": "2015-09-30T23:27:55.768Z"
        }
```

```
    ]
}
```

2. `create-configuration-template` で環境の現在の設定を保存する

```
$ aws elasticbeanstalk create-configuration-template --environment-id e-vcghmm2zwk
 --application-name my-app --template-name v1
```

Elastic Beanstalk は、Amazon S3 の Elastic Beanstalk バケットに設定を保存します。

JSON ドキュメント

を使用して環境 AWS CLI を作成および更新する場合は、JSON 形式で設定オプションを指定することもできます。JSON の設定ファイルのライブラリは、 を使用して環境を作成および管理する場合 AWS CLI に便利です。

たとえば、次の JSON ドキュメントはアプリケーションのヘルスチェック URL を `/health` に設定します。

~/ebconfigs/healthcheckurl.json

```
[
  {
    "Namespace": "aws:elasticbeanstalk:application",
    "OptionName": "Application Healthcheck URL",
    "Value": "/health"
  }
]
```

EB CLI 設定

EB CLI では、保存済み設定と eb config コマンドによる直接的な環境設定がサポートされるだけでなく、環境内のインスタンスへの SSH アクセスのための Amazon EC2 キーペアを指定するのに使用できる `default_ec2_keyname` という名前の設定ファイルを利用できます。EB CLI は、このオプションを使用して `EC2KeyName` 名前空間に `aws:autoscaling:launchconfiguration` 設定オプションを設定します。

~/workspace/my-app/.elasticbeanstalk/config.yml

```
branch-defaults:
  master:
```

```
    environment: my-env
  develop:
    environment: my-env-dev
deploy:
  artifact: ROOT.war
global:
  application_name: my-app
  default_ec2_keyname: my-keypair
  default_platform: Tomcat 8 Java 8
  default_region: us-west-2
  profile: null
  sc: git
```

環境の作成時の設定オプションの設定

Elastic Beanstalk コンソール、EB CLI、 SDK AWS CLI、または Elastic Beanstalk API を使用して AWS Elastic Beanstalk 環境を作成する場合、環境とその中で起動される AWS リソースをカスタマイズするための設定オプションの値を指定できます。

設定変更が 1 回限りでない場合は、ソースバンドルまたは Amazon S3 にローカルに設定ファイルを保存できます。

このトピックでは、環境の作成時に設定オプションを設定するすべての方法について、具体的手順を説明します。

クライアント

- Elastic Beanstalk コンソールで
- EB CLI の使用
- の使用 AWS CLI

Elastic Beanstalk コンソールで

Elastic Beanstalk コンソールで Elastic Beanstalk 環境を作成する場合は、設定ファイル、保存済み設定、および [Create New Environment (新しい環境の作成)] ウィザード内のフォームを使用して設定オプションを指定できます。

メソッド

- 設定ファイルの使用 (.ebextensions)
- 保存済み設定を使用する

- 新しい環境のウィザードを使用する

設定ファイルの使用 (**.ebextensions**)

.config という名前のフォルダ内のアプリケーションソースバンドルに .ebextensions ファイルを含めます。

設定ファイルの詳細については、「.Ebextensions」を参照してください。

```
~/workspace/my-app-v1.zip
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- index.php
`-- styles.css
```

環境の作成中に、通常ソースバンドルを Elastic Beanstalk にアップロードします。

Elastic Beanstalk コンソールは、一部の設定オプションに推奨値を適用し、その他のオプションをフォームフィールドで指定します。Elastic Beanstalk コンソールで設定されるオプションは、環境に直接適用され、設定ファイルの設定を上書きします。

保存済み設定を使用する

Elastic Beanstalk コンソールを使って新しい環境を作成する場合、最初の手順の 1 つとして設定を選択します。設定は、[PHP] や [Tomcat] などのプラットフォームの最新バージョンに代表される [事前定義の設定] である場合や、[保存された設定]である場合があります。

環境の作成時に保存された設定を適用するには (Elastic Beanstalk コンソール)

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. ナビゲーションペインで、アプリケーションの名前を見つけ、[保存された設定] を選択します。
4. 適用する保存済み設定を選択し、[Launch environment (環境の起動)] を選択します。
5. ウィザードを続行して環境を作成します。

保存された設定はアプリケーション固有です。保存済み設定を作成する方法の詳細については、「保存された設定」を参照してください。

新しい環境のウィザードを使用する

標準設定オプションのほとんどは、ウィザードの [構成の詳細] ページと [アクセス許可] ページに表示されます。環境の Amazon RDS データベースを作成するか、VPC を設定する場合、これらのリソースで追加の設定オプションを利用できます。

環境の作成時に設定オプションを設定するには (Elastic Beanstalk コンソール)、

1. Elastic Beanstalk 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、[アプリケーション] を選択します。
3. アプリケーションを選択または作成します。
4. [アクション]、[環境の作成] の順に選択します。
5. ウィザードを進めて、[さらにオプションを設定] を選択します。
6. [設定のプリセット] のいずれかを選択し、1 つ以上の設定カテゴリで [編集] を選択して、関連する設定オプションのグループを変更します。
7. オプションの選択が完了したら、[環境の作成] を選択します。

新しい環境のウィザードで設定したオプションは、環境に直接適用され、保存済み設定や適用する設定ファイル (`.ebextensions`) のオプション設定を上書きします。環境の作成後に EB CLI や AWS CLI を使用して設定を削除すると、保存済み設定や設定ファイルの設定を有効にすることができます。

新しい環境のウィザードの詳細については、「新しい環境の作成ウィザード」を参照してください。

EB CLI の使用

メソッド

- 設定ファイルの使用 (.ebextensions)
- 保存済み設定を使用する
- コマンドラインオプションを使用する

設定ファイルの使用 (**`.ebextensions`**)

`.config` の下のプロジェクトフォルダに `.ebextensions` ファイルを含め、アプリケーションコードとともにこれらをデプロイします。

設定ファイルの詳細については、「.Ebextensions」を参照してください。

```
~/workspace/my-app/
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- .elasticbeanstalk
|   `-- config.yml
|-- index.php
`-- styles.css
```

環境を作成し、eb create で環境にソースコードをデプロイします。

```
~/workspace/my-app$ eb create my-env
```

保存済み設定を使用する

eb create で環境を作成する際に保存済み設定を適用するには、`--cfg` オプションを使用します。

```
~/workspace/my-app$ eb create --cfg savedconfig
```

保存した設定は、プロジェクトフォルダまたは Amazon S3 の Elastic Beanstalk ストレージの場所に保存できます。上記の例では、EB CLI は最初にフォルダ `savedconfig.cfg.yml` で保存された設定ファイル `.elasticbeanstalk/saved_configs/` を検索します。`.cfg.yml` で保存済み設定を適用する際は、ファイル名拡張子 (`--cfg`) を含めないでください。

```
~/workspace/my-app/
|-- .ebextensions
|   `-- healthcheckurl.config
|-- .elasticbeanstalk
|   |-- saved_configs
|   |   `-- savedconfig.cfg.yml
|   `-- config.yml
|-- index.php
`-- styles.css
```

EB CLI がローカルに設定を見つけられない場合、Amazon S3 の Elastic Beanstalk ストレージの場所を検索します。保存された構成の作成、編集、およびアップロードの詳細については、を参照してください保存された設定

コマンドラインオプションを使用する

EB CLI eb create コマンドには、環境の作成時にオプションを設定する目的で使用できる複数のオプションがあります。これらのオプションを使用すると、RDS データベースを環境に追加する、VPC を設定する、または推奨値を上書きすることができます。

たとえば、EB CLI はデフォルトで `t2.micro` インスタンスタイプを使用します。別のインスタンスタイプを選択するには、`--instance_type` オプションを使用します。

```
$ eb create my-env --instance_type t2.medium
```

Amazon RDS データベースインスタンスを作成して環境にアタッチするには、`--database` オプションを使用します。

```
$ eb create --database.engine postgres --database.username dbuser
```

環境の名前やデータベースパスワードなどの環境の作成に必要なパスワードを省略すると、EB CLI はこれらの入力を求めます。

利用可能なオプションのリストと使用例については、eb create を参照してください。

の使用 AWS CLI

`create-environment` コマンドを使用して で Elastic Beanstalk 環境を作成する場合 AWS CLI、AWS CLI は推奨値を適用しません。指定するソースバンドルの設定ファイルで定義されているすべての設定オプション

メソッド

- 設定ファイルの使用 (.ebextensions)
- 保存済み設定を使用する
- コマンドラインオプションを使用する

設定ファイルの使用 (`.ebextensions`)

で作成した環境に設定ファイルを適用するには AWS CLI、Amazon S3 にアップロードするアプリケーションソースバンドルにファイルを含めます。

設定ファイルの詳細については、「.Ebextensions」を参照してください。

```
~/workspace/my-app-v1.zip
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- index.php
`-- styles.css
```

アプリケーションソースバンドルをアップロードし、 を使用して環境を作成するには AWS CLI

1. Amazon S3 にまだ Elastic Beanstalk バケットがない場合は、`create-storage-location` でバケットを作成します。

```
$ aws elasticbeanstalk create-storage-location
{
    "S3Bucket": "elasticbeanstalk-us-west-2-123456789012"
}
```

2. アプリケーションソースバンドルを Amazon S3 にアップロードします。

```
$ aws s3 cp sourcebundle.zip s3://elasticbeanstalk-us-west-2-123456789012/my-app/
sourcebundle.zip
```

3. アプリケーションバージョンを作成します。

```
$ aws elasticbeanstalk create-application-version --application-name my-app --
version-label v1 --description MyAppv1 --source-bundle S3Bucket="elasticbeanstalk-
us-west-2-123456789012",S3Key="my-app/sourcebundle.zip" --auto-create-application
```

4. 環境を作成します。

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-
name my-env --version-label v1 --solution-stack-name "64bit Amazon Linux 2015.03
 v2.0.0 running Tomcat 8 Java 8"
```

保存済み設定を使用する

保存済み設定を環境の作成時に適用するには、`--template-name` パラメータを使用します。

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-name
 my-env --template-name savedconfig --version-label v1
```

保存済み設定を指定する際は、同時にソリューションスタック名を指定しないでください。保存済み設定は既にソリューションスタックを指定しているため、両方のオプションを使用しようとすると Elastic Beanstalk はエラーを返します。

コマンドラインオプションを使用する

`--option-settings` パラメータを使用して JSON 形式で設定オプションを使用します。

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-name
 my-env --version-label v1 --template-name savedconfig --option-settings '[
  {
    "Namespace": "aws:elasticbeanstalk:application",
    "OptionName": "Application Healthcheck URL",
    "Value": "/health"
  }
]
```

ファイルから JSON をロードするには、`file://` プレフィックスを使用します。

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-
name my-env --version-label v1 --template-name savedconfig --option-settings file://
healthcheckurl.json
```

Elastic Beanstalk は、`--option-settings` オプションで指定されるオプション設定を環境に直接適用します。保存済み設定や設定ファイルで同じオプションが指定されている場合、`--option-settings` はこれらの値を上書きします。

環境の作成後に設定オプションを設定する

保存済み設定を適用し、新しいソースバンドルと設定ファイル (`.ebextensions`) または JSON ドキュメントをアップロードすることで、実行中の環境でオプション設定を変更できます。EB CLI および Elastic Beanstalk コンソールには、オプション設定の設定と更新を実行するためのクライアント固有の機能もあります。

設定オプションを設定または変更する場合、変更の重要度によっては環境全体の更新をトリガーできます。例えば、`aws:autoscaling:launchconfiguration` のオプションへの変更（`InstanceType` など）では、環境内の Amazon EC2 インスタンスの再プロビジョニングが必要です。これにより、ローリング更新がトリガーされます。他の設定変更は、中断や再プロビジョニングなしに適用されます。

EB CLI または AWS CLI コマンドを使用して、環境からオプション設定を削除できます。API レベルで環境に直接設定されたオプションを削除すると、これらの設定にマスキングされていた設定ファイルの設定が有効になります。

保存済み設定と設定ファイルの設定は、他の設定方法のいずれかを使用して環境に直接同じオプションを設定することで上書きできます。ただし、これらは、更新された保存済み設定ファイルまたは設定ファイルを適用することによってのみ完全に削除することができます。オプションが、保存済み設定や設定ファイルに設定されていない場合、または環境に直接設定されていない場合、デフォルト値が適用されます (存在する場合)。詳細については、「優先順位」を参照してください。

クライアント

- Elastic Beanstalk コンソール
- EB CLI
- AWS CLI「××

Elastic Beanstalk コンソール

設定ファイルを含むアプリケーションソースバンドルをデプロイする、保存済み設定を適用する、または環境マネジメントコンソールの [Configuration (設定)] ページで直接変更することにより、Elastic Beanstalk コンソールの設定オプションの設定を更新することができます。

メソッド

- 設定ファイルの使用 (.ebextensions)
- 保存済み設定を使用する
- Elastic Beanstalk コンソールを開きます。

設定ファイルの使用 (`.ebextensions`)

ソース ディレクトリにある設定ファイルを更新し、新しいソースバンドルを作成し、変更を適用する Elastic Beanstalk 環境に新しいバージョンをデプロイします。

設定ファイルの詳細については、「.Ebextensions」を参照してください。

ソースバンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。

3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

設定ファイルへの変更は、保存済み設定や API レベルで環境に直接適用された設定のオプション設定を上書きしません。詳細については、「優先順位」を参照してください。

保存済み設定を使用する

実行中の環境に保存済み設定を適用し、同設定に定義されたオプション設定を適用します。

保存された設定を実行環境に適用するには (Elastic Beanstalk コンソール)、

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. ナビゲーションペインで、アプリケーションの名前を見つけ、[保存された設定] を選択します。
4. 適用する保存済み設定を選択し、[Load (ロード)] を選択します。
5. 環境を選択した後、[ロード] を選択します。

保存済み設定で定義された設定は、設定ファイルの設定を上書きし、環境マネジメントコンソールを使用して設定された設定に上書きされます。

保存済み設定を作成する方法の詳細については、「保存された設定」を参照してください。

Elastic Beanstalk コンソールを開きます。

Elastic Beanstalk コンソールでは、各環境の [Configuration (設定)] ページに多くの設定オプションが表示されます。

実行環境の設定オプションを変更するには (Elastic Beanstalk コンソール)

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. 編集する設定ページを見つけます。

- 目的のオプションを見つけたか、そのオプションが含まれている設定カテゴリがわかったら、設定カテゴリの [編集] を選択します。
- オプションを探すには、[テーブルビュー] をオンにし、検索ボックスに検索語句を入力します。入力中に、リストが短くなり、検索語句に一致したオプションのみが表示されます。

  目的のオプションを見つけたら、そのオプションが含まれている設定カテゴリの [編集] を選択します。

5. 設定を変更し、次に [保存] を選択します。
6. 必要に応じて、追加の設定カテゴリで前の 2 つの手順を繰り返します。
7. [Apply] を選択します。

環境マネジメントコンソールの設定オプションへの変更は、環境に直接適用されます。これらの変更により、設定ファイルまたは保存済み設定の同じオプションの設定が上書きされます。詳細については、「優先順位」を参照してください。

Elastic Beanstalk コンソールを使用して実行環境の設定オプションを変更する方法の詳細については、「Elastic Beanstalk 環境の設定」のトピックを参照してください。

EB CLI

設定ファイルを含むソースコードをデプロイする、保存済み設定の設定を適用する、または eb config コマンドで環境設定を直接変更することにより、設定オプションの設定を EB CLI で更新できます。

メソッド

- 設定ファイルの使用 (.ebextensions)
- 保存済み設定を使用する
- eb config を使用する
- eb setenv を使用する

設定ファイルの使用 (`.ebextensions`)

`.config` の下のプロジェクトフォルダに `.ebextensions` ファイルを含め、アプリケーションコードとともにこれらをデプロイします。

設定ファイルの詳細については、「.Ebextensions」を参照してください。

```
~/workspace/my-app/
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- .elasticbeanstalk
|   `-- config.yml
|-- index.php
`-- styles.css
```

eb deploy でソースコードをデプロイします。

```
~/workspace/my-app$ eb deploy
```

保存済み設定を使用する

eb config コマンドを使用して、実行中の環境に保存済み設定を適用できます。保存済み設定の名前で `--cfg` オプションを使用し、その設定を環境に適用できます。

```
$ eb config --cfg v1
```

この例の [v1] は、以前に作成された保存済みファイルの名前です。

このコマンドで環境に適用された設定は、環境の作成時に適用された設定やアプリケーションソースバンドルの設定ファイルで定義された設定を上書きします。

eb config を使用する

EB CLI の eb config コマンドにより、テキストエディタを使用して環境で直接オプション設定を設定および削除できます。

eb config を実行している場合、EB CLI は、設定ファイル、保存済み設定、推奨値、環境に直接設定されたオプション、API デフォルトなどのすべてのソースから環境に適用されている設定を表示します。

> **Note**
>
> eb config は、環境プロパティを表示しません。アプリケーションで読み取り可能な環境プロパティを設定するには、eb setenv を使用します。

次の例では、`aws:autoscaling:launchconfiguration` 名前空間に適用される設定を示します。設定は次のとおりです。

- 2 つの推奨値、`IamInstanceProfile` および `InstanceType` は、環境作成時に EB CLI によって適用されます。
- オプション `EC2KeyName` は、リポジトリ設定に基づいて作成時に環境に直接適用されます。
- その他のオプションの API デフォルト値。

```
ApplicationName: tomcat
DateUpdated: 2015-09-30 22:51:07+00:00
EnvironmentName: tomcat
SolutionStackName: 64bit Amazon Linux 2015.03 v2.0.1 running Tomcat 8 Java 8
settings:
...
aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: null
    EC2KeyName: my-key
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
    ImageId: ami-1f316660
    InstanceType: t2.micro
...
```

eb config で設定オプションを設定するまたは変更するには

1. eb config を実行して環境の設定を表示します。

   ```
   ~/workspace/my-app/$ eb config
   ```

2. デフォルトのテキストエディタを使用して任意の設定値をへんこうする。

   ```
   aws:autoscaling:launchconfiguration:
       BlockDeviceMappings: null
       EC2KeyName: my-key
       IamInstanceProfile: aws-elasticbeanstalk-ec2-role
       ImageId: ami-1f316660
       InstanceType: t2.medium
   ```

3. 一時設定ファイルを保存して終了します。
4. EB CLI が環境設定を更新します。

eb config で設定オプションを設定すると、他のすべてのソースの設定が上書きされます。

eb config で環境からオプションを削除することもできます。

設定オプションを削除するには（EB CLI）

1. eb config を実行して環境の設定を表示します。

```
~/workspace/my-app/$ eb config
```

2. 表示された値を文字列 null で置き換えます。削除するオプションを含む行全体を削除することもできます。

```
aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: null
    EC2KeyName: my-key
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
    ImageId: ami-1f316660
    InstanceType: null
```

3. 一時設定ファイルを保存して終了します。
4. EB CLI が環境設定を更新します。

eb config で環境からオプションを削除すると、アプリケーションソースバンドルの設定ファイルの同じオプションの設定が有効になります。詳細については、「優先順位」を参照してください。

eb setenv を使用する

EB CLI で環境プロパティを設定するには、eb setenv を使用します。

```
~/workspace/my-app/$ eb setenv ENVVAR=TEST
INFO: Environment update is starting.
INFO: Updating environment my-env's configuration settings.
INFO: Environment health has transitioned from Ok to Info. Command is executing on all
 instances.
INFO: Successfully deployed new configuration to environment.
```

このコマンドは、aws:elasticbeanstalk:application:environment 名前空間の環境プロパティを設定します。eb setenv で設定された環境プロパティは、短い更新プロセスの後、アプリケーションで利用できます。

環境に設定された環境プロパティを eb printenv で表示します。

```
~/workspace/my-app/$ eb printenv
 Environment Variables:
     ENVVAR = TEST
```

AWS CLI「►►

で設定オプション設定を更新するには、設定ファイルを含むソースバンドルをデプロイ AWS CLI するか、リモートで保存された設定を適用するか、 `aws elasticbeanstalk update-environment` コマンドを使用して環境を直接変更します。

方法

- 設定ファイルの使用 (.ebextensions)
- 保存済み設定を使用する
- コマンドラインオプションを使用する

設定ファイルの使用 (`.ebextensions`)

で実行中の環境に設定ファイルを適用するには AWS CLI、Amazon S3 にアップロードするアプリケーションソースバンドルにファイルを含めます。

設定ファイルの詳細については、「.Ebextensions」を参照してください。

```
~/workspace/my-app-v1.zip
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- index.php
`-- styles.css
```

アプリケーションソースバンドルをアップロードし、実行中の環境に適用するには (AWS CLI)

1. Amazon S3 にまだ Elastic Beanstalk バケットがない場合は、`create-storage-location` でバケットを作成します。

   ```
   $ aws elasticbeanstalk create-storage-location
   {
       "S3Bucket": "elasticbeanstalk-us-west-2-123456789012"
   ```

```
}
```

2. アプリケーションソースバンドルを Amazon S3 にアップロードします。

```
$ aws s3 cp sourcebundlev2.zip s3://elasticbeanstalk-us-west-2-123456789012/my-app/
sourcebundlev2.zip
```

3. アプリケーションバージョンを作成します。

```
$ aws elasticbeanstalk create-application-version --application-
name my-app --version-label v2 --description MyAppv2 --source-bundle
 S3Bucket="elasticbeanstalk-us-west-2-123456789012",S3Key="my-app/
sourcebundlev2.zip"
```

4. 環境を更新します。

```
$ aws elasticbeanstalk update-environment --environment-name my-env --version-label
 v2
```

保存済み設定を使用する

`--template-name` コマンドの `aws elasticbeanstalk update-environment` オプションにより、実行中の環境に保存済み設定を適用できます。

保存された設定は、`resources/templates`下のアプリケーションの名前を付けたパスのElastic Beanstalk バケットにある必要があります。例えば、US West (オレゴン) リージョン (us-west-2) の `v1` アプリケーション用の `my-app` テンプレート (アカウント番号 123456789012) は `s3://elasticbeanstalk-us-west-2-123456789012/resources/templates/my-app/v1` にあります。

実行中の環境に保存済み設定を適用するには (AWS CLI)

- `update-environment` オプションで、`--template-name` コールの保存済み設定を指定します。

```
$ aws elasticbeanstalk update-environment --environment-name my-env --template-
name v1
```

Elastic Beanstalk は、`aws elasticbeanstalk create-configuration-template` を使用して設定を作成すると、保存された設定をこの場所に配置します。保存済み設定をローカルに変更し、自身でこの場所に配置することもできます。

コマンドラインオプションを使用する

JSON ドキュメントで設定オプションを変更するには (AWS CLI)

1. ローカルファイルで、オプション設定を JSON 形式で定義します。
2. `update-environment` オプションで `--option-settings` を実行します。

```
$ aws elasticbeanstalk update-environment --environment-name my-env --option-
settings file://~/ebconfigs/as-zero.json
```

この例では、ゼロインスタンスの最小数と最大数で環境を設定するオプションを [`as-zero.json`] が定義します。これにより、環境を終了することなく、インスタンスが停止します。

~/ebconfigs/as-zero.json

```
[
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MinSize",
        "Value": "0"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MaxSize",
        "Value": "0"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "RollingUpdateEnabled",
        "Value": "false"
    }
]
```

> **Note**
>
> `update-environment` で設定オプションを設定すると、他のすべてのソースの設定が上書きされます。

`update-environment` で環境からオプションを削除することもできます。

設定オプションを削除するには（AWS CLI）

- `update-environment` オプションを使用して `--options-to-remove` コマンドを実行します。

```
$ aws elasticbeanstalk update-environment --environment-name my-env --options-to-remove Namespace=aws:autoscaling:launchconfiguration,OptionName=InstanceType
```

`update-environment` で環境からオプションを削除すると、アプリケーションソースバンドルの設定ファイルの同じオプションの設定が有効になります。これらの方法でオプションが設定されている場合、API のデフォルト値が適用されます（存在する場合）。詳細については、「優先順位」を参照してください。

すべての環境に対する汎用オプション

名前空間

- aws:autoscaling:asg
- aws:autoscaling:launchconfiguration
- aws:autoscaling:scheduledaction
- aws:autoscaling:trigger
- aws:autoscaling:updatepolicy:rollingupdate
- aws:ec2:instances
- aws:ec2:vpc
- aws:elasticbeanstalk:application
- aws:elasticbeanstalk:application:environment
- aws:elasticbeanstalk:application:environmentsecrets
- aws:elasticbeanstalk:cloudwatch:logs

- aws:elasticbeanstalk:cloudwatch:logs:health
- aws:elasticbeanstalk:command
- aws:elasticbeanstalk:environment
- aws:elasticbeanstalk:environment:process:default
- aws:elasticbeanstalk:environment:process:process_name
- aws:elasticbeanstalk:environment:proxy:staticfiles
- aws:elasticbeanstalk:healthreporting:system
- aws:elasticbeanstalk:hostmanager
- aws:elasticbeanstalk:managedactions
- aws:elasticbeanstalk:managedactions:platformupdate
- aws:elasticbeanstalk:monitoring
- aws:elasticbeanstalk:sns:topics
- aws:elasticbeanstalk:sqsd
- aws:elasticbeanstalk:trafficsplitting
- aws:elasticbeanstalk:xray
- aws:elb:healthcheck
- aws:elb:loadbalancer
- aws:elb:listener
- aws:elb:listener:listener_port
- aws:elb:policies
- aws:elb:policies:policy_name
- aws:elbv2:listener:default
- aws:elbv2:listener:listener_port
- aws:elbv2:listenerrule:rule_name
- aws:elbv2:loadbalancer
- aws:rds:dbinstance

aws:autoscaling:asg

環境の Auto Scaling グループを設定します。詳細については、「the section called “Auto Scaling グループ”」を参照してください。

名前空間: **`aws:autoscaling:asg`**

名前	説明	デフォルト	有効な値

Availability Zones	アベイラビリティーゾーン (AZs) は、AWS リージョン内の個別の場所であり、他の AZs の障害から分離されるように設計されています。同一リージョン内にあるアベイラビリティーゾーン間には、低価格かつ低レイテンシーのネットワーク接続が提供されます。インスタンスの AZ の数を選択します。	`Any`	`Any`
			`Any 1`
			`Any 2`
			`Any 3`

Cooldown	クールダウンの期間を指定すると、前のアクティビティの効果が表示される前に、Amazon EC2 Auto Scaling が追加のスケーリングアクティビティを開始するのを防ぐのに役立ちます。クールダウン期間とはスケーリングアクティビティが完了してから別のスケーリングアクティビティが開始されるまでの時間（秒単位）です。	`360`	`0 ~ 10000`

Custom Availability Zones	インスタンスの AZ を定義します。	なし	`us-east-1a`
			`us-east-1b`
			`us-east-1c`
			`us-east-1d`
			`us-east-1e`
			`eu-central-1`

EnableCapacityRebalancing	Auto Scaling グループのスポットインスタンスに対して容量の再調整機能を有効にするかどうかを指定します。詳細については、Amazon EC2 Auto	`false`	`true` `false`

名前	説明	デフォルト	有効な値
	Scaling ユーザーガイドの「容量の再調整」を参照してください。 このオプションは、`aws:ec2:instances` 名前空間で `EnableSpot` が `true` に設定されており、Auto Scaling グループに少なくとも 1 つのスポットインスタンスがある場合にのみ関連します。		
MinSize	Auto Scaling グループで使用するインスタンスの最小数。	`1`	`1 ~ 10000`
MaxSize	Auto Scaling グループで使用するインスタンスの最大数。	`4`	`1 ~ 10000`

aws:autoscaling:launchconfiguration

環境の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを設定します。

環境に使用されるインスタンスは、Amazon EC2 起動テンプレートまたは Auto Scaling グループ起動設定リソースを使用して作成されます。以下のオプションは、両方のリソースタイプで使用できます。

詳細については、「the section called “Amazon EC2 インスタンス”」を参照してください。Amazon Elastic Block Store (EBS) の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EBS」の章を参照してください。

名前空間: **`aws:autoscaling:launchconfiguration`**

名前	説明	デフォルト	有効な値

DisableDefaultEC2SecurityGroup	デフォルト値の に設定すると`false`、Elastic Beanstalk は、HTTP (80) の標準ポートでインターネットまたはロードバランサーからのトラフィックを許可するデフォルトのセ	`false`	`true`
			
			`false`

名前	説明	デフォルト	有効な値
	キュリティグループを作成します。このセキュリティグループは、環境の作成時に環境の EC2 インスタンスにアタッチされます。 に設定すると`true`、Elastic Beanstalk は新しい環境の EC2 インスタンスにデフォルトのセキュリティグループを割り当てません。既存の環境の場合、Elastic Beanstalk は環境の EC2 インスタンスからデフォルトの EC2 セキュリティグループの割り当てを解除します。そのため、次の設定も設定する必要があります。 • この名前空間`SecurityGroups` のオプションでは、カスタムセキュリティグループ (複数可) を定義するために少なくとも 1 つの値が必要です。 • ロードバランサーを使用する環境では、別の名前空間`SecurityGroups` のオプションを設定して、ロードバランサーのカスタムセキュリティグループを設定する必要があります。アプリケーションロードバランサーの場合は、 aws:elbv2:loadbalancer名前空間で オプションを設定します。クラシックロードバランサーの場合は、 aws:elb:loadbalancer名前空間で オプションを設定します。 • 詳細については、「EC2 セキュリティグループの管理」を参照してください。		

名前	説明	デフォルト	有効な値
	がtrueデフォルトのセキュリティグループにDisableDefaultEC2SecurityGroup　設定された環境EC2KeyNameの に値が指定されている場合、EC2 インスタンスには関連付けられません。		

名前	説明	デフォルト	有効な値

DisableIMDSv1	インスタンスメタデータサービスバージョン 1 (IMDSv1) を無効にして IMDSv2 を適用するには、`true` に設定します。
	
	IMDSv1 と IMDSv2 の両方を有効にするには、`false` に設定します。
	
	環境のインスタンスのデフォルトは、プラットフォームのオペレーティングシステムに基づいて次のようになります。
	
	• Windows サーバー、AL2 以前 – IMDSv1 と IMDSv2 の両方が有効になります (DisableIMDSv1 のデフォルトは `false`)
	• AL2023 – IMDSv2 のみが有効になります (DisableIMDSv1 のデフォルトは `true`)
	
	詳細については、「インスタンスメタデータサービスの設定」を参照してください。
	
	Important
	このオプションを設定すると、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポ
		`false` – Windows サーバー、Amazon Linux 2 以前をベースとするプラットフォーム
		
		`true` – Amazon Linux 2023 ベースのプラットフォーム
			`true`
			
			`false`

名前	説明	デフォルト	有効な値
	リシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してください[Elastic Beanstalk 環境を起動テンプレートに移行する](#)。		
EC2KeyName	EC2 インスタンスには、キーペアを使用して安全なログインができます。 が`true`デフォルトのセキュリティグループに`DisableDefaultEC2SecurityGroup` 設定された環境`EC2KeyName` の に値が指定されている場合、EC2 インスタンスには関連付けられません。 **Note** Elastic Beanstalk コンソールを使用して環境を作成する場合は、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールによって、推奨値に上書きされます。	なし	

名前	説明	デフォルト	有効な値
IamInstanceProfile	インスタンスプロファイルにより、AWS Identity and Access Management (IAM) ユーザーおよび AWS サービスは一時的なセキュリティ認証情報にアクセスして AWS API コールを行うことができます。インスタンスプロファイルの名前または ARN を指定します。 例: • `aws-elasticbeanstalk-ec2-role` • `arn:aws:iam::123456789012:instance-profile/aws-elasticbeanstalk-ec2-role` **Note** Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。	なし	インスタンスプロファイル名または ARN。
ImageId	独自のカスタム AMI ID を指定して、デフォルトの Amazon Machine Image (AMI) を上書きすることができます。 例: `ami-1f316660`	なし	

名前	説明	デフォルト	有効な値
InstanceType	Elastic Beanstalk 環境でアプリケーションを実行するために使用されるインスタンスタイプ。 **Important** `InstanceType` オプションはサポートされなくなりました。`InstanceTypes` 名前空間のより新しく強力な `aws:ec2:instances` オプションに置き換わりました。この新しいオプションを使用して、環境で 1 つ以上のインスタンスタイプのリストを指定することができます。このリストの最初の値は、ここで説明する `aws:autoscaling:launchconfiguration` 名前空間に含まれる `InstanceType` オプションの値に等しくなります。新しいオプションを使用してインスタンスタイプを指定することをお勧めします。指定した場合、新しいオプションは古いオプションよりも優先されます。詳細については、「the section called “aws:ec2:instances 名前空間”」を参照してください。 使用可能なインスタンスタイプは、使用されるアベイラビリティーゾーンとリージョンによって異なります。サブ	アカウントとリージョンによって異なります。	1 つの EC2 インスタンスタイプ。 アカウント、リージョン、アベイラビリティーゾーンによって異なります。これらの値でフィルタリングされた Amazon EC2 インスタンスタイプのリストを取得できます。詳細については、「Amazon EC2 ユーザーガイド」の「利用可能なインスタンスタイプ」を参照してください。

名前	説明	デフォルト	有効な値
	ネットを選択した場合、そのサブネットを含むアベイラビリティーゾーンによって、使用可能なインスタンスタイプが決まります。 • Elastic Beanstalk は、Amazon EC2 Mac インスタンスタイプを support していません。 • Amazon EC2 インスタンスファミリーとタイプの詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスタイプ」を参照してください。 • 各リージョンで利用可能なインスタンスタイプの詳細については、「Amazon EC2 ユーザーガイド」の「利用可能なインスタンスタイプ」を参照してください。 Note Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。		

名前	説明	デフォルト	有効な値

LaunchTemplateTagPropagationEnabled	`true` に設定すると、環境にプロビジョニングされた特定のリソースの起動テンプレートへの環境タグの伝播が可能になります。
	
	Elastic Beanstalk は、次のリソースの起動テンプレートにのみタグを伝播できます。
	
	• EBS ボリューム
	• EC2インスタンス
	• EC2 ネットワークインターフェイス
	• AWS CloudFormation リソースを定義する起動テンプレート
	
	CloudFormation では特定のリソースのテンプレート作成時にのみタグが許可されるため、この制約が存在します。詳細については、「AWS CloudFormation ユーザーガイド」の「TagSpecification」を参照してください。
	
	Important
	
	• 既存の環境でこのオプション値を `false` から `true` に変更すると、既存のタグに互換性を破る変更が生じる可能性があります。
	• この機能を有効にすると、タグの伝播に EC2 の置き換えが必要になり、ダウンタイムが発生する可能性があります。ローリング更新を有効に
		`false`	`true`
			
			`false`

名前	説明	デフォルト	有効な値
	して設定の変更を一括で適用し、更新プロセス中のダウンタイムを防ぐことができます。詳細については、「設定変更」を参照してください。 起動テンプレートの詳細については、以下を参照してください。 • 「Amazon EC2 Auto Scaling ユーザーガイド」の「起動テンプレート」 • 「AWS CloudFormation ユーザーガイド」の「テンプレートの操作」 • 「AWS CloudFormation ユーザーガイド」の「Elastic Beanstalk テンプレートスニペット」 このオプションの詳細については、「起動テンプレートへのタグの伝播」を参照してください。		

MonitoringInterval	Amazon CloudWatch メトリクスが返される間隔 (分単位)。	`5 minute`	`1 minute`
			
			`5 minute`

名前	説明	デフォルト	有効な値
SecurityGroups	Auto Scaling グループ内の EC2 インスタンスに割り当てて、インスタンスのファイアウォールルールを定義するために使用する、Amazon EC2 セキュリティグループ ID を一覧表示します。 このオプションを と共に使用`DisableDefaultEC2SecurityGroup` して、EC2 インスタンスのファイアウォールルールを定義する独自のカスタムセキュリティグループをアタッチします。詳細については、「負荷分散された (マルチインスタンス) 環境」を参照してください。 **Important** EC2 インスタンスへの着信トラフィックがブロックされないように、追加の設定を完了する必要がある場合があります。これは、カスタム EC2 セキュリティグループを持つマルチインスタンス環境にのみ適用されます。EC2 セキュリティグループには、ロードバランサーからルーティングされたトラフィックへのアクセスを許可するインバウンドルールを含める必要があります。詳細については、「マルチインスタンス環境での EC2 セキュリティグループの管理」を参照してください。	`elasticbeanstalk-default`	

名前	説明	デフォルト	有効な値
	既存の Amazon EC2 セキュリティグループの ID や、テンプレートで作成した AWS::EC2::SecurityGroup リソースへの参照を含む値をカンマで区切り、1 つの文字列として指定できます。 この名前空間の が に設定されている場合、このオプション`DisableDefaultEC2SecurityGroup` には少なくとも 1 つの値を指定する必要があります`true`。		

名前	説明	デフォルト	有効な値
SSHSource Restriction	環境に対する SSH アクセスをロックするために使用します。例えば、プライベートサブネット内のインスタンスには、拠点ホストからのみアクセスが可能になるように、EC2 インスタンスに対する SSH アクセスをロックできます。 この文字列は次のような形式になります。 *protocol*, *fromPort*, *toPort*, *source_restriction* *protocol* 進入ルールのプロトコル。 *fromPort* 開始ポート番号。 *toPort* 終了ポート番号。 *source_restriction* Classless Inter-Domain Routing (CIDR) 範囲、またはトラフィックのルートが経由する必要があるセキュリティグループ。セキュリティグループ ID を使用してセキュリティグループを指定します。 別のアカウントからセキュリティグループを指定するには、セキュリティグループ ID の前に AWS アカウント ID をスラッシュで区切	なし	

名前	説明	デフォルト	有効な値
	って含めます。もう 1 つのアカウントは同じ AWS リージョンに存在する必要があります。構文は *aws-account-id* /*security-group-id* であることに注意してください。例: **123456789012 /sg-99999999** 例: • `tcp, 22, 22, 54.240.196.185/32` • `tcp, 22, 22, my-security-group-id` • `tcp, 22, 22, 123456789012/their-security-group-id`		

名前	説明	デフォルト	有効な値
BlockDeviceMappings	Auto Scaling グループ内のすべてのインスタンスに追加の Amazon EBS ボリュームまたはインスタンスストアボリュームをアタッチします。 **Important** このオプションを設定すると、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してください Elastic Beanstalk 環境を起動テンプレートに移行する 。 インスタンスストアボリュームをマッピングする場合は、ボリューム名にデバイス名をマップする必要があるだけです。ただし、Amazon EBS ボリュームをマッピングする場合は、次のフィールドの一部またはすべてを追加で指定することをお勧めします (各	なし	• サイズ — 500 ~ 16384 GiB の範囲内でなければなりません • スループット — 1 秒あたり 125 ~ 1000 メビバイト (MiB/s) でなければなりません

名前	説明	デフォルト	有効な値
	フィールドはコロンで区切る必要があります)。 • スナップショット ID • サイズ (GB 単位) • 合わせて削除 (`true` または `false`) • ストレージタイプ (`gp3`、`gp2`、`standard`、`st1`、`sc1`、または `io1` の場合のみ) • IOPS (`gp3` または `io1` の場合のみ) • スループット (`gp3` の場合のみ) 次の例では、3 つの Amazon EBS ボリューム (1 つ (空) の 100 GB gp2 ボリュームと 1 つのスナップショット、1 つ (空) の 20 GB の io1 ボリュームと 2000 個のプロビジョンド IOPS、およびインスタンスストアボリューム `ephemeral0`) をアタッチします。インスタンスタイプでサポートされている場合は、複数のインスタンスストアボリュームを添付することができます。 `/dev/sdj=:100:true:gp2,/dev/sdh=snap-51eef269,/dev/sdi=:20:true:io1:2000,/dev/sdb=ephemeral0`		

名前	説明	デフォルト	有効な値
RootVolumeType	環境の EC2 インスタンスにアタッチされた、Amazon EBS のルートボリューム用のボリュームタイプ (マグネティック、汎用 SSD、プロビジョンド IOPS SSD)。 **Important** このオプションを設定すると、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してください Elastic Beanstalk 環境を起動テンプレートに移行する 。	タイプはプラットフォームによって異なります。	マグネティックストレージの場合、`standard`。 汎用 SSD の場合、`gp2` または `gp3`。 プロビジョンド IOPS SSD の場合、`io1`。

名前	説明	デフォルト	有効な値
RootVolumeSize	Amazon EBS ルートボリュームのストレージ容量を総 GB で表示します。 `RootVolumeType` をプロビジョンド IOPS SSDに設定した場合は必須。 例えば、`"64"`。	マグネティックストレージおよび汎用 SSD の場合はプラットフォームごとに異なります。 プロビジョンド IOPS SSD の場合はありません。	汎用とプロビジョンド IOPS SSD の場合は `10` から `16384` GB。 マグネティックの場合は 8 から `1024` GB。
RootVolumeIOPS	プロビジョンド IOPS SSD ルートボリュームまたは汎用 gp3 SSD ルートボリュームに適用する、1 秒あたりの入力/出力オペレーション (IOPS) 数。 IOPS とボリュームサイズとの比率は最大で 500:1 です。たとえば、3000 IOPS のボリュームのサイズは 6 GiB 以上である必要があります。	なし	io1 プロビジョンド IOPS SSD ボリュームの場合、`100` から `20000`。 汎用 gp3 SSD ルートボリュームの場合、`3000` から `16000`。
RootVolumeThroughput	環境の EC2 インスタンスにアタッチされた Amazon EBS ルートボリュームのプロビジョニングに必要なスループット。1 秒あたりのメガバイト数 (MiB/s) で示します。 **Note** このオプションは、gp3 ストレージタイプにのみ適用されます。	なし	`125` ~ `1000`

aws:autoscaling:scheduledaction

環境の Auto Scaling グループのスケジュールされたアクションを設定します。アクションごとに、オプション名、名前空間、各設定の値に加え、`resource_name` を指定します。例については、「aws:autoscaling:scheduledaction 名前空間」を参照してください。

名前空間: **`aws:autoscaling:scheduledaction`**

名前	説明	デフォルト	有効な値
StartTime	ワンタイムアクションの場合は、アクションを実行する日時を選択します。繰り返しアクションの場合は、アクションをアクティブ化する日時を選択します。	なし	スケジュールに基づくすべてのスケーリングアクションで一意な ISO-8601 タイムスタンプ。
EndTime	スケジュールに基づくスケーリングアクションの繰り返しを停止する将来の日時 (UTC/GMT タイムゾーン)。[EndTime] を指定しない場合、アクションは `Recurrence` 式に基づいて繰り返されます。 例: `2015-04-28T04:07:2Z` スケジュールに基づくアクションが終了しても、Amazon EC2 Auto Scaling は自動的に以前の設定に戻りません。必要に応じて、元の設定に戻るように、スケジュールに基づいて 2 つ目のアクションを設定します。	なし	スケジュールに基づくすべてのスケーリングアクションで一意な ISO-8601 タイムスタンプ。
MaxSize	アクション実行時に適用される最大インスタンス数。	なし	`0` ~ `10000`
MinSize	アクション実行時に適用される最小インスタンス数。	なし	`0` ~ `10000`

名前	説明	デフォルト	有効な値
DesiredCapacity	Auto Scaling グループの初期に必要な容量を設定します。スケジュールに基づくアクションが適用されたら、トリガーは設定に基づき、希望する容量を調整します。	なし	`0` ~ `10000`
Recurrence	スケジュールに基づきアクションを実行する頻度。繰り返しを指定していない場合、スケーリングアクションは、`StartTime` に指定されたタイミングで1 回だけ実行されます。	なし	Cron 式。
Suspend	スケジュールされた繰り返しアクションを一時的に無効にするには `true` に設定します。	`false`	`true` `false`

aws:autoscaling:trigger

環境の Auto Scaling グループのスケーリングトリガーを設定します。

> **Note**
>
> この名前空間の 3 つのオプションによって、定義されている上限をトリガーのメトリクスが超えている時間がどれぐらい続くとトリガーが起動されるか決まります。これらのオプションは次のように関連しています。
>
> `BreachDuration = Period * EvaluationPeriods`
>
> これらのオプションのデフォルト値 (それぞれ 5、5、1) は、この式を満たします。不整合な値を指定すると、引き続き式が満たされるように、Elastic Beanstalk によっていずれかの値が変更される場合があります。

名前空間: `aws:autoscaling:trigger`

名前	説明	デフォルト	有効な値
BreachDuration	トリガーが発生する前に、定義された制限 (`UpperThreshold` と `LowerThreshold` で指定される値) をメトリクスが超えることができる時間 (分単位)。	`5`	`1 ~ 600`
LowerBreachScaleIncrement	スケーリングアクティビティを実行するときに削除できる Amazon EC2 インスタンス数。	`-1`	
LowerThreshold	測定値がこの数未満になった状態が超過期間だけ続くと、トリガーが発生します。	`2000000`	`0 ~ 20000000`

MeasureName	Auto Scaling をトリガーするために使用されるメトリクス。
	
	Note
	`HealthyHostCount` 、`UnhealthyHostCount` 、および `TargetResponseTime` は、専用のロードバランサーを使用している環境にのみ適用されます。これらメトリクス値は、共有ロードバランサーで構成された環境では無効です。ロードバランサータイプの詳細については、「Elastic Beanstalk 環境のロードバランサー」を参照してください。
		`NetworkOut`	`CPUUtilization`
			
			`NetworkIn`
			
			`NetworkOut`
			
			`DiskWriteOps`
			
			`DiskReadBytes`
			
			`DiskReadOps`
			
			`DiskWriteBytes`
			
			`Latency`
			
			`RequestCount`

名前	説明	デフォルト	有効な値

			`HealthyHostCount`
			`UnhealthyHostCount`
			`TargetResponseTime`

Period	Amazon CloudWatch でトリガーのメトリクスを測定する頻度を指定します。値は、2 つの連続する期間の分数です。	`5`	`1 ~ 600`
EvaluationPeriods	違反が発生しているかどうかを判断するために使用される連続した評価期間の回数。	`1`	`1 ~ 600`

Statistic	トリガーに使用する `Average` などの統計データ。	`Average`	`Minimum`
			`Maximum`
			`Sum`
			`Average`

名前	説明	デフォルト	有効な値
Unit	トリガー用の測定単位。Bytes など。	Bytes	Seconds Percent Bytes Bits Count Bytes/Second Bits/Second Count/Second None
UpperBreachScaleIncrement	スケーリングアクティビティを実行するときに追加すべき Amazon EC2 インスタンス数を指定します。	1	
UpperThreshold	測定値がこの数を超えた状態が超過期間だけ続くと、トリガーが発生します。	6000000	0 ~ 20000000

aws:autoscaling:updatepolicy:rollingupdate

環境の Auto Scaling グループのローリング更新を設定します。

名前空間: **`aws:autoscaling:updatepolicy:rollingupdate`**

名前	説明	デフォルト	有効な値
MaxBatchSize	ローリング更新の各バッチに含まれるインスタンス数。	Auto Scaling グループ内の最小サイズの 3 分の 1 で	1 ~ 10000

名前	説明	デフォルト	有効な値
		次に大きい整数に丸められます。	
MinInstancesInService	他のインスタンスが終了する間、Auto Scaling グループ内で使用中となる必要があるインスタンスの最小数。	Auto Scaling グループの最小サイズ、または Auto Scaling グループの最大サイズよりひとつ下のサイズのいずれか小さい方。	0 ~ 9999

名前	説明	デフォルト	有効な値
RollingUpdateEnabled	true の場合、環境のローリング更新が可能になります。ローリング更新は、Elastic Beanstalk ソフトウェアアプリケーションに小規模な更新を頻繁に実行する必要があり、アプリケーションのダウンタイムを発生させないようにする場合に役立ちます。 この値を true に設定すると、自動的に MaxBatchSize 、MinInstancesInService 、および PauseTime オプションが有効になります。また、これらのオプションのいずれかを設定すると、自動的に RollingUpdateEnabled オプションの値が true に設定されます。このオプションを false に設定すると、ローリング更新が無効になります。	false	true false

名前	説明	デフォルト	有効な値
	Note Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。		

名前	説明	デフォルト	有効な値

RollingUpdateType	更新には、時間ベースのローリング更新、正常性ベースのローリング更新、イミュータブルな更新の 3 種類があります。
	
	時間ベースのローリング更新では、バッチとバッチの間に PauseTime が適用されます。ヘルスベースのローリング更新では、 新しいインスタンスがヘルスチェックにパスしてはじめて次のバッチに移ります。イミュータブルな更新では、新しい Auto Scaling グループでフルセットのインスタンスが起動されます。
	
	Note
	Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオ
		`Time`	`Time`
			
			`Health`
			
			`Immutable`

名前	説明	デフォルト	有効な値
	プションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。		
PauseTime	Elastic Beanstalk サービスが、インスタンスの 1 つのバッチの更新を完了してから次のバッチに取り掛かるまで待機する時間 (秒、分、または時間単位)。	インスタンスタイプとコンテナに基づいて自動的に計算されます。	PT0S* (0 秒) ~ PT1H (1 時間)
Timeout	1 つのバッチ内のすべてのインスタンスがヘルスチェックにパスするまでの最大待機時間 (分または時間単位)。これを過ぎると更新がキャンセルされます。	PT30M (30 分)	PT5M* (5 分) ~ PT1H (1 時間) *ISO8601 日付形式で、PT#H#M#S のようにする必要があります。それぞれ # は時間数、分数、秒数を示します。

aws:ec2:instances

スポットオプションを含め、環境のインスタンスを設定します。この名前空間は、`aws:autoscaling:launchconfiguration` と `aws:autoscaling:asg` を補完します。

詳細については、「the section called “Auto Scaling グループ”」を参照してください。

名前空間: **`aws:ec2:instances`**

名前	説明	デフォルト	有効な値

EnableSpot	環境のスポットインスタンスリクエストを有効にします。`false` の場合、この名前空間の一部のオプションが有効になりません。
	
	Important
	このオプションを設定すると、Elastic Beanstalk は起動設定を使用して既存の環境を起動テンプレートに移行できます。これを行うには、起動テンプレートを管理するために必要なアクセス許可が必要です。これらのアクセス許可は、 管理ポリシーに含まれています。管理ポリシーの代わりにカスタムポリシーを使用すると、環境設定を更新すると、環境の作成または更新が失敗する可能性があります。詳細およびその他の考慮事項については、「」を参照してくださいElastic Beanstalk 環境を起動テンプレートに移行する 。
		`false`	`true`
			
			`false`

InstanceTypes	環境で使用するインスタンスタイプのコンマ区切りのリスト (例: `t2.micro,t3.micro`)。	2 つのインス	1 ~ 40 個の EC2 インスタンスタイプ。少なくと

名前	説明	デフォルト	有効な値
	EnableSpot が trueで、SpotAllocationStrategy が に設定されている場合capacity-optimized-prioritized 、このオプションで指定された値のリストによって、スポットインスタンス割り当て戦略のインスタンスタイプの優先度が決まります。 スポットインスタンスがアクティブ化していない場合 (EnableSpot は false)、リストの最初のインスタンスタイプのみが使用されます。 このオプションのリストの最初のインスタンスタイプは、InstanceType 名前空間の aws:autoscaling:launchconfiguration オプションの値と同じです。後者のオプションは古い形式なため、使用はお勧めしません。両方を指定した場合、InstanceTypes オプションのリストの最初のインスタンスタイプが使用され、InstanceType は無視されます。 使用可能なインスタンスタイプは、使用されるアベイラビリティーゾーンとリージョンによって異なります。サブネットを選択した場合、そのサブネットを含むアベイラビリティーゾーンによって、使用可能なインスタンスタイプが決まります。	インスタンスタイプのリスト。 アカウントとリージョンによって異なります。	も 2 つをレコメンデーションします アカウント、リージョン、アベイラビリティーゾーンによって異なります。これらの値でフィルタリングされた Amazon EC2 インスタンスタイプのリストを取得できます。詳細については、「Amazon EC2 ユーザーガイド」の「利用可能なインスタンスタイプ」を参照してください。 インスタンスタイプは、すべて同じアーキテクチャに属している必要があります (arm64,x86_64,i386)。 Supported Architectures もこの名前空間の一部です。に任意の値を指定した場合Supported Architectures の場合、入力する値 InstanceTypes 提供しているアーキテクチャのうちの 1 つのみに属している必要があります。Supported Architectures 。

名前	説明	デフォルト	有効な値
	• Elastic Beanstalk は、Amazon EC2 Mac インスタンスタイプを support していません。 • Amazon EC2 インスタンスファミリーとタイプの詳細については、「Amazon EC2 ユーザーガイド」の「インスタンスタイプ」を参照してください。 • 各リージョンで利用可能なインスタンスタイプの詳細については、「Amazon EC2 ユーザーガイド」の「利用可能なインスタンスタイプ」を参照してください。 Note 一部の古い AWS アカウントでは、スポットインスタンスをサポートしないデフォルトのインスタンスタイプ (t1.micro など) が Elastic Beanstalk に提供される場合があります。スポットインスタンス・リクエストをアクティベーションして、スポットを support していないインスタンスタイプに関するエラーが表示される場合は、スポットを support するインスタンスタイプを設定してください。スポットインスタンスタイプを選択するには、スポットイン		

名前	説明	デフォルト	有効な値
	スタンスアドバイザーを使用します。 環境設定を更新し、InstanceTypes オプションから 1 つ以上のインスタンスタイプを削除すると、Elastic Beanstalk は、削除されたインスタンスタイプで実行されているすべての Amazon EC2 インスタンスを終了します。次に、環境の Auto Scaling グループは、現在指定されているインスタンスタイプを使用して、必要に応じて新しいインスタンスを起動し、必要な容量を完了します。		
SpotAllocationStrategy	使用可能なスポットキャパシティプールからスポットインスタンスを割り当てる方法を決定するスポットインスタンス割り当て戦略を指定します。 に設定するとcapacity-optimized-prioritized 、 の値の順序によって、割り当てのインスタンスタイプの優先度InstanceTypes が設定されます。 このオプションは、EnableSpot が true の場合にのみ関連します。	capacity-optimized	capacity-optimized price-capacity-optimized capacity-optimized-prioritized lowest-price

名前	説明	デフォルト	有効な値
SpotFleet OnDemandBase	環境のスケールアップ時にスポットインスタンスを考慮する前に、Auto Scaling グループがプロビジョニングするオンデマンドインスタンスの最小数。 このオプションは、`EnableSpot` が `true` の場合にのみ関連します。	`0`	`0` 名前空間の `MaxSize` ~ `aws:autoscaling:asg` オプション
SpotFleet OnDemandAboveBasePercentage	Auto Scaling グループが `SpotOnDemandBase` インスタンスを超えてプロビジョニングする追加容量の一部としてのオンデマンドインスタンスの割合。 このオプションは、`EnableSpot` が `true` の場合にのみ関連します。	単一インスタンス環境の `0` 負荷分散された環境の `70`	`0` ~ `100`

名前	説明	デフォルト	有効な値

SpotMaxPrice	スポットインスタンスに対する支払いが許容される、単位時間あたりの上限料金 (USD)。スポットインスタンスの上限価格オプションに関する推奨事項については、「Amazon EC2 ユーザーガイド」の「スポットインスタンスの料金履歴」を参照してください。
	
	このオプションは、`EnableSpot` が `true` の場合にのみ関連します。
		各インスタンスタイプのオンデマンド価格。この場合のオプションの値は `null` です。	`0.001 ~ 20.0`
			
			`null`

名前	説明	デフォルト	有効な値

SupportedArchitectures	環境で使用する Amazon EC2 インスタンスアーキテクチャタイプのコンマ区切りリスト。
	
	Elastic Beanstalk は、次のプロセッサアーキテクチャに基づいてインスタンスタイプをサポートします。
	
	• AWS Graviton 64 ビット Arm アーキテクチャ (arm64)
	• 64 ビットアーキテクチャ (x86_64)
	• 32ビットアーキテクチャ (i386)
	
	プロセッサアーキテクチャと Amazon EC2 インスタンスタイプの詳細については、「」を参照してください。the section called “Amazon EC2 インスタンスタイプ”。
		なし	`arm64`
			
			`x86_64`
			
			`i386`
			
			Note
			32 ビットアーキテクチャ`i386`は、大部分の Elastic Beanstalk プラットフォームではサポートされていません。選択することをお勧めします。`x86_64`または`arm64`代わりにアーキテクチャタイプ。

aws:ec2:vpc

カスタム Amazon Virtual Private Cloud (Amazon VPC) でリソースを起動するように環境を設定します。この名前空間で設定を行わないと、Elastic Beanstalk はデフォルトの VPC でリソースを起動します。

名前空間: **`aws:ec2:vpc`**

名前	説明	デフォルト	有効な値
VPCId	Amazon VPC の ID。	なし	
Subnets	Auto Scaling グループのサブネットまたはサブネットの ID。複数のサブネットがある場合は、各サブネット ID を単一のカンマで区切った文字列 (例: `"subnet-11111111,subnet-22222222"`)として値を指定します。	なし	
ELBSubnets	elastic load balancer の単一あるいは複数のサブネットの ID。複数のサブネットがある場合は、各サブネット ID を単一のカンマで区切った文字列 (例: `"subnet-11111111,subnet-22222222"`)として値を指定します。	なし	
ELBScheme	Amazon VPC に内部ロードバランサーを作成して、Amazon VPC の外部から Elastic Beanstalk アプリケーションにアクセスできないようにする場合は `internal` を指定します。`public` または `internal` 以外の値を指定すると、Elastic Beanstalk はその値を無視します。	`public`	`public` `internal`
DBSubnets	データベースサブネットの ID が含まれています。Amazon RDS DB インスタンスをアプリケーションの一部として追加する場合にのみ使用します。複数のサブネットがある場合は、各サブネット ID を単一のカンマで区切った文字列 (例: `"subnet-11111111,subnet-22222222"`)として値を指定します。	なし	
AssociatePublicIpAddress	Amazon VPC 内のパブリック IP アドレスを持つインスタンスを起動するかどうかを指定します。パブリック IP アドレスを持つインスタンスは、インターネットと通信するために NAT デバイスを必要としません。ロードバランサーとインスタンスを 1 つのパブリック	なし	`true` `false`

名前	説明	デフォルト	有効な値
	サブネットに含める場合は、値を `true` に設定する必要があります。 このオプションは単一インスタンス環境には効果がありません。この環境には Elastic IP アドレスがある単一の Amazon EC2 インスタンスが必ずあります。このオプションは、負荷分散されたスケーラブルな環境に関連します。		

aws:elasticbeanstalk:application

アプリケーションのヘルスチェックパスを設定します。詳細については、「ベーシックヘルスレポート」を参照してください。

名前空間: **`aws:elasticbeanstalk:application`**

名前	説明	デフォルト	有効な値
Application Healthcheck URL	ヘルスチェックのリクエストの送信先となるパス。このパスが設定されていない場合、ロードバランサーはポート 80 で TCP 接続を確立し、アプリケーションのヘルスステータスの検証を試みます。/ で始まるパスに設定し、HTTP GET リクエストをそのパスに送信します。プロトコル (HTTP、HTTPS、TCP、SSL) およびパスの前のポートを含めて、HTTPS 接続を確認することも、デフォルトでないポートを使用することもできます。	なし	有効な値を次に示します。 / (HTTP GET からルートパス) */health* `HTTPS:443/` `HTTPS:443/` *`health`*

名前	説明	デフォルト	有効な値
	Note Elastic Beanstalk コンソールを使用して環境を作成する場合は、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールによって、推奨値に上書きされます。		

EB CLI および Elastic Beanstalk コンソールでは、上記のオプションに推奨値が適用されます。設定ファイルを使用して同じファイルを設定する場合は、これらの設定を削除する必要があります。詳細については、「推奨値」を参照してください。

aws:elasticbeanstalk:application:environment

アプリケーションの環境プロパティを設定します。

名前空間: **`aws:elasticbeanstalk:application:environment`**

名前	説明	デフォルト	有効な値
環境変数の名前。	キーと値のペアを渡します。	なし	環境変数の値。

詳細については「環境変数およびその他のソフトウェア設定」を参照してください。

aws:elasticbeanstalk:application:environmentsecrets

アプリケーションの環境シークレットとして機能するように環境変数を設定します。環境シークレットは、 AWS Secrets Manager シークレットまたは Parameter Store AWS Systems Manager パラメータを保存します。

名前空間: **`aws:elasticbeanstalk:application:environmentsecrets`**

名前	説明	デフォルト	有効な値
OptionName	シークレットストアまたはパラメータストアの値を保持する環境変数の名前を指定します。	なし	環境変数の名前。
値	AWS Secrets Manager または Parameter Store に保存されている値の ARN AWS Systems Manager を指定します。インスタンスのブートストラップ中に、Elastic Beanstalk は環境変数をこの ARN リソースに保存されている値に開始します。 **Note** 環境の EC2 インスタンスプロファイルロールがシークレットとパラメータ ARNs にアクセスするために必要なアクセス許可が設定されていることを確認します。詳細については、「必要な IAM 許可」を参照してください。	なし	AWS Secrets Manager シークレットまたは Parameter Store AWS Systems Manager パラメータ値の有効な ARN 値。

詳細については、「シークレットを環境変数として設定する」を参照してください。

aws:elasticbeanstalk:cloudwatch:logs

アプリケーションのインスタンスログストリーミングを設定します。

名前空間: **`aws:elasticbeanstalk:cloudwatch:logs`**

名前	説明	デフォルト	有効な値

StreamLogs	プロキシおよびデプロイログ用に CloudWatch Logs でグループを作成し、環境の各インスタンスからログをストリーミングするかどうかを指定します。	`false`	`true`
			
			`false`

DeleteOnTerminate	環境が終了したときにロググループを削除するかどうかを指定します。`false` の場合、ログは `RetentionInDays` 日間保持されます。	`false`	`true`
			
			`false`

RetentionInDays	有効期限が切れるまでログイベントを保持する日数。	7	1、3、5、7、14、30、60、90、120、150、180、365、400、545、731、1827、3653

aws:elasticbeanstalk:cloudwatch:logs:health

アプリケーションの環境ヘルスのログストリーミングを設定します。

名前空間: **`aws:elasticbeanstalk:cloudwatch:logs:health`**

名前	説明	デフォルト	有効な値

HealthStreamingEnabled	拡張ヘルスレポートが有効になっている環境で、環境ヘルスの CloudWatch Logs でグループを作成し、Elastic Beanstalk 環境ヘルスデータをアーカイブするかどうかを指定します。拡張ヘルスを有効にする方法については、`aws:elasticbeanstalk:healthreporting:system` を参照してください。	`false`	`true`
			
			`false`

名前	説明	デフォルト	有効な値

DeleteOnTerminate	環境が終了したときにロググループを削除するかどうかを指定します。`false` の場合、ヘルスデータは `RetentionInDays` 日間保持されます。	`false`	`true`
			
			`false`

RetentionInDays	有効期限が切れる前にアーカイブされたヘルスデータを保持する日数。	7	1、3、5、7、14、30、60、90、120、150、180、365、400、545、731、1827、3653

aws:elasticbeanstalk:command

アプリケーションコードのデプロイポリシーを設定します。詳細については、「the section called “デプロイオプション”」を参照してください。

名前空間: **`aws:elasticbeanstalk:command`**

名前	説明	デフォルト	有効な値

DeploymentPolicy	Deployment policy を選択しアプリケーションバージョンをデプロイします。
	
	Note
	Elastic Beanstalk コンソールを使用して環境を作成する場合は、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールによって、推奨値に上書きされます。
		`AllAtOnce`	`AllAtOnce`
			
			`Rolling`
			
			`RollingWithAdditionalBatch`
			
			`Immutable`
			
			`TrafficSplitting`

名前	説明	デフォルト	有効な値
Timeout	インスタンスでコマンドの実行が完了するまでの待機時間 (秒)。 Elastic Beanstalk は、内部的に 240 秒 (4 分) を `Timeout` 値に加算します。たとえば、効果的なタイムアウトはデフォルトでは 840 秒 (600+240)、または 14 分です。	`600`	`1 ~ 3600`

BatchSizeType	BatchSize で指定される数値のタイプ。
	
	Note
	Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。
		`Percentage`	`Percentage`
			
			`Fixed`

BatchSize	同時にデプロイを実行する、Auto Scaling グループ内の Amazon EC2 インスタンスの割合または一定数。有効な値は、BatchSizeType の設定によって異なります。 **Note** Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。	`100`	`1 ~ 100` (`Percentage`)。 `1` (aws:autoscaling:asg::MaxSize (`Fixed`) に)

名前	説明	デフォルト	有効な値
IgnoreHealthCheck	ヘルスチェックに合格しなくてもデプロイをキャンセルしません。	`false`	`true` `false`

aws:elasticbeanstalk:environment

環境のアーキテクチャとサービスのロールを設定します。

名前空間: **`aws:elasticbeanstalk:environment`**

名前	説明	デフォルト	有効な値
EnvironmentType	`SingleInstance` に設定すると、ロードバランサーなしの EC2 インスタンスが 1 つ起動されます。	`LoadBalanced`	`SingleInstance` `LoadBalanced`
ServiceRole	Elastic Beanstalk が環境のリソースを管理するために使用する IAM ロールの名前。ロール名 (オプションでカスタムパスを付けたプレフィックス) またはその ARN を指定します。 例: • `aws-elasticbeanstalk-service-role` • *`custom-path /custom-role`* • `arn:aws:iam::123456789012:role/aws-elasticbeanstalk-service-role`	なし	IAM ロール名、パス/名前、または ARN

名前	説明	デフォルト	有効な値
	Note Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。		

LoadBalancerType	環境のロードバランサーのタイプ。詳細については、「the section called “ロードバランサー”」を参照してください。	`classic`	`classic`
			`application`
			`network`

LoadBalancerIsShared	環境のロードバランサーが専用か共有かを指定します。このオプションは、Application Load Balancer に対してのみ設定できます。環境が作成されてから後で変更することはできません。 `false` の場合、環境には独自の専用ロードバランサーがあり、Elastic Beanstalk によって作成および管理されます。 の場合、`true`環境はユーザーが作成し、aws:elbv2:loadbancer 名前空間の `SharedLoadBalancer` オプションで指定した共有ロードバランサーを使用します。	`false`	`true` `false`

aws:elasticbeanstalk:environment:process:default

環境のデフォルトのプロセスを設定します。

名前空間: `aws:elasticbeanstalk:environment:process:default`

名前	説明	デフォルト	有効な値
DeregistrationDelay	登録を解除する前の、アクティブなリクエストが完了するまでの待機時間 (秒単位)。	20	0 ~ 3600
HealthCheckInterval	Elastic Load Balancing が、アプリケーションの Amazon EC2 インスタンスでヘルスチェックを行う時間間隔 (秒単位)。	標準ロードバランサーまたはアプリケーションロードバランサー: 15 network load balancer: 30	標準ロードバランサーまたはアプリケーションロードバランサー: 5 から 300 network load balancer: 10、30
HealthCheckPath	ヘルスチェックの HTTP リクエストが送信されるパス。	/	ルーティング可能なパス。
HealthCheckTimeout	ヘルスチェック中に、その応答を待つ時間 (秒単位)。 このオプションは、アプリケーションロードバランサーを使用する環境にのみ適用されます。	5	1 ~ 60
HealthyThresholdCount	Elastic Load Balancing がインスタンスのヘルスステータスを変更するまでの間に、連続して成功したリクエスト数。	標準ロードバランサーまたはアプリケーションロードバランサー: 3 network load balancer: 5	2 ~ 10

名前	説明	デフォルト	有効な値
MatcherHTTPCode	インスタンスが正常であることを示す HTTP コードのカンマ区切りのリスト。 このオプションは、ネットワークまたは Application Load Balancer を使用する環境にのみ適用されます。	200	Application Load Balancer: 200 ~ 499 Network Load Balancer: 200 ~ 399
Port	プロセスがリッスンするポート。	80	1 ~ 65535
Protocol	プロセスで使用するプロトコル。 アプリケーションロードバランサーを使用する場合は、このオプションを HTTP または HTTPS にのみ設定できます。 network load balancer を使用する場合は、このオプションを TCP にのみ設定できます。	標準ロードバランサーまたはアプリケーションロードバランサー: HTTP network load balancer: TCP	TCP HTTP HTTPS

名前	説明	デフォルト	有効な値

StickinessEnabled	スティッキーセッションを有効にするには、true に設定します。
	
	このオプションは、アプリケーションロードバランサーを使用する環境にのみ適用されます。
		`'false'`	`'false'`
			
			`'true'`

StickinessLBCookieDuration	スティッキーセッション Cookie の有効期間 (秒単位)。 このオプションは、アプリケーションロードバランサーを使用する環境にのみ適用されます。	`86400` (1 日)	`1 ~ 604800`
StickinessType	スティッキーセッション用の Cookie を使用するには、`lb_cookie` に設定します。 このオプションは、アプリケーションロードバランサーを使用する環境にのみ適用されます。	`lb_cookie`	`lb_cookie`

名前	説明	デフォルト	有効な値
UnhealthyThresholdCount	Elastic Load Balancing がインスタンスのヘルスステータスを変更するまでの間に、連続して失敗したリクエスト数。	5	2 ~ 10

aws:elasticbeanstalk:environment:process:process_name

環境に応じて追加のプロセスを設定します。

名前空間: **`aws:elasticbeanstalk:environment:process:`*`process_name`***

名前	説明	デフォルト	有効な値
DeregistrationDelay	登録を解除する前の、アクティブなリクエストが完了するまでの待機時間 (秒単位)。	20	0 ~ 3600
HealthCheckInterval	Elastic Load Balancing が、アプリケーションの Amazon EC2 インスタンスでヘルスチェックを行う間隔 (秒単位)。	標準ロードバランサーまたはアプリケーションロードバランサー: 15 network load balancer: 30	標準ロードバランサーまたはアプリケーションロードバランサー: 5 から 300 network load balancer: 10、30
HealthCheckPath	ヘルスチェックの HTTP リクエストが送信されるパス。	/	ルーティング可能なパス。

名前	説明	デフォルト	有効な値
HealthCheckTimeout	ヘルスチェック中に、その応答を待つ時間 (秒単位)。 このオプションは、アプリケーションロードバランサーを使用する環境にのみ適用されます。	5	1 ~ 60
HealthyThresholdCount	Elastic Load Balancing がインスタンスのヘルスステータスを変更するまでの間に、連続して成功したリクエスト数。	標準ロードバランサーまたはアプリケーションロードバランサー: 3 network load balancer: 5	2 ~ 10
MatcherHTTPCode	インスタンスが正常であることを示す HTTP コードのリスト (カンマ区切り)。 このオプションは、ネットワークまたは Application Load Balancer を使用する環境にのみ適用されます。	200	Application Load Balancer: 200 ~ 499 Network Load Balancer: 200 ~ 399
Port	プロセスがリッスンするポート。	80	1 ~ 65535

名前	説明	デフォルト	有効な値
Protocol	プロセスで使用するプロトコル。 アプリケーションロードバランサーを使用する場合は、このオプションを HTTP または HTTPS にのみ設定できます。 network load balancer を使用する場合は、このオプションを TCP にのみ設定できます。	標準ロードバランサーまたはアプリケーションロードバランサー: HTTP network load balancer: TCP	TCP HTTP HTTPS
StickinessEnabled	スティッキーセッションを有効にするには、true に設定します。 このオプションは、アプリケーションロードバランサーを使用する環境にのみ適用されます。	'false'	'false' 'true'
StickinessLBCookieDuration	スティッキーセッション Cookie の有効期間 (秒単位)。 このオプションは、アプリケーションロードバランサーを使用する環境にのみ適用されます。	86400 (1 日)	1 ~ 604800

名前	説明	デフォルト	有効な値
StickinessType	スティッキーセッション用の Cookie を使用するには、`lb_cookie` に設定します。 このオプションは、アプリケーションロードバランサーを使用する環境にのみ適用されます。	`lb_cookie`	`lb_cookie`
UnhealthyThresholdCount	Elastic Load Balancing がインスタンスのヘルスステータスを変更するまでの間に、連続して失敗したリクエスト数。	`5`	`2 ~ 10`

aws:elasticbeanstalk:environment:proxy:staticfiles

静的ファイルを処理するようにプロキシサーバーを設定するには、次の名前空間を使用できます。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接処理します。これにより、アプリケーションで処理する必要があるリクエストの数が減ります。

プロキシサーバーで処理されるパスを、静的アセットを含むソースコード内のフォルダにマッピングします。この名前空間で定義される各オプションは、それぞれ異なるパスをマッピングします。

> **Note**
>
> この名前空間は、Amazon Linux 2 以降に基づくプラットフォームブランチに適用されます。ご使用の環境で Amazon Linux AMI (Amazon Linux 2より前) に基づくプラットフォームバー

ジョンを使用している場合、プラットフォーム固有の静的ファイル名前空間については the section called “プラットフォーム固有のオプション” を参照してください。

名前空間: **`aws:elasticbeanstalk:environment:proxy:staticfiles`**

名前	値
プロキシサーバーのファイルの保存先となるパス。値の先頭は / にします。 例えば、`/images` を指定して *`subdomain`* `.eleasticbeanstalk.com/images` でファイルを配信します。	ファイルを含むフォルダの名前。 例えば、`staticimages` を指定してソースバンドルの最上位にある `staticimages` という名前のフォルダのファイルを配信します。

aws:elasticbeanstalk:healthreporting:system

環境に合わせて拡張ヘルスレポートを設定します。

名前空間: **`aws:elasticbeanstalk:healthreporting:system`**

名前	説明	デフォルト	有効な値

SystemType	ヘルスレポートシステム (ベーシックまたは拡張)。拡張ヘルスレポートは、サービスロールとバージョン 2 以降のプラットフォームのバージョンを必要とします。
	
	Note
	Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。
		`basic`	`basic`
			
			`enhanced`

名前	説明	デフォルト	有効な値
ConfigDocument	CloudWatch に発行する環境およびインスタンスメトリクスを定義した JSON ドキュメント。	なし	

EnhancedHealthAuthEnabled	Elastic Beanstalk が、環境インスタンスから Elastic Beanstalk サービスに向けて、拡張ヘルス情報を通信するために使用する内部 API の認可を有効にします。
	
	詳細については、「the section called “拡張ヘルスレポートのロール”」を参照してください。
	
	Note
	このオプションは、拡張ヘルスレポート (`SystemType` が `enhanced` に設定されている場合など) にのみ適用できます。
		`true`	`true`
			
			`false`

HealthCheckSuccessThreshold	インスタンスがヘルスチェックにパスするようにしきい値を下げます。
	
	Note
	Elastic Beanstalk コンソールを使用して環境を作成する場合は、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールによって、推奨値に上書きされます。
		`Ok`	`Ok`
			
			`Warning`
			
			`Degraded`
			
			`Severe`

aws:elasticbeanstalk:hostmanager

更新したログを Amazon S3 にアップロードするように、環境内の EC2 インスタンスを設定します。

名前空間: **`aws:elasticbeanstalk:hostmanager`**

名前	説明	デフォルト	有効な値

LogPublicationControl	アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーします。	`false`	`true`
			
			`false`

aws:elasticbeanstalk:managedactions

ご使用の環境に合わせて管理対象プラットフォームの更新を設定します。

名前空間: **`aws:elasticbeanstalk:managedactions`**

名前	説明	デフォルト	有効な値

ManagedActionsEnabled	マネージドプラットフォーム更新を有効にします。
	
	これを `true` に設定する場合は、`PreferredStartTime` と `UpdateLevel` も指定する必要があります。
		`false`	`true`
			
			`false`

PreferredStartTime	マネージドアクションのメンテナンス時間を UTC で設定します。 例えば、`"Tue:09:00"` 。	なし	曜日と時間を *##:##:#* の形式で設定。
ServiceRoleForManagedUpdates	Elastic Beanstalk が環境のマネージドプラットフォーム更新を実行するために使用する IAM ロールの名前。	なし	`ServiceRole` と同じ または `AWSServiceRoleForE`

名前	説明	デフォルト	有効な値
	`ServiceRole` 名前空間の `aws:elasticbeanstalk:environment` オプションに指定したものと同じロールを使用することも、アカウントのマネージド更新サービスにリンクされたロールを使用することもできます。後者の場合、マネージド更新サービスにリンクされたロールがまだアカウントになければ、Elastic Beanstalk によって作成されます。		`lasticBeanstalkManagedUpdates`

aws:elasticbeanstalk:managedactions:platformupdate

ご使用の環境に合わせて管理対象プラットフォームの更新を設定します。

名前空間: **`aws:elasticbeanstalk:managedactions:platformupdate`**

名前	説明	デフォルト	有効な値
UpdateLevel	マネージドプラットフォーム更新で適用される最高レベルの更新。プラットフォームのバージョンの表記は、`####.#####.###` です。たとえば、2.0.8 はメジャーバージョン 2、マイナーバージョン 0、パッチ バージョン 8 であることを意味します。	なし	パッチバージョンのみ更新する場合は `patch`。 マイナーバージョンとパッチバージョンの両方を更新する場合は `minor`。

InstanceRefreshEnabled	毎週のインスタンス置換を有効にします。	`false`	`true`
			
			`false`

名前	説明	デフォルト	有効な値
	`ManagedActionsEnabled` を `true`に設定する必要があります。		

aws:elasticbeanstalk:monitoring

ヘルスチェックに失敗した EC2 インスタンスを削除するように環境を設定します。

名前空間: **`aws:elasticbeanstalk:monitoring`**

名前	説明	デフォルト	有効な値

Automatically Terminate Unhealthy Instances	ヘルスチェックに失敗した場合にインスタンスを終了します。
	
	Note
	このオプションは、レガシー環境でのみサポートされていました。これおよび他のインスタンスベースのメトリクスに到達できるかどうかに基づいてインスタンスの状態を決定します。Elastic Beanstalk では、アプリケーションの状態に基づいて、自動的にインスタンスを終了させる方法はありません。
		`true`	`true`
			
			`false`

aws:elasticbeanstalk:sns:topics

ご使用の環境の通知を設定します。

名前空間: **`aws:elasticbeanstalk:sns:topics`**

名前	説明	デフォルト	有効な値
Notification Endpoint	アプリケーションに影響する重要なイベントについて通知を受け取るエンドポイント。 **Note** Elastic Beanstalk コンソールを使用して環境を作成する場合は、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールによって、推奨値に上書きされます。	なし	

Notification Protocol	エンドポイントに通知を送信するために使用するプロトコル。	`email`	`http`
			
			`https`
			
			`email`
			
			`email-json`
			
			`sqs`

Notification Topic ARN	サブスクライブするトピックの Amazon リソースネーム (ARN)。	なし	
Notification Topic Name	サブスクライブするトピックの名前。	なし	

aws:elasticbeanstalk:sqsd

Amazon SQS キューをワーカー環境用に設定します。

名前空間: **`aws:elasticbeanstalk:sqsd`**

名前	説明	デフォルト	有効な値
WorkerQueueURL	ワーカー環境枠のデーモンがメッセージを読み込むキューの URL **Note** 値を指定しない場合、Elastic Beanstalk が自動的に作成するキューは標準の Amazon SQS キューになります。値を指定すると、標準または FIFO Amazon SQS キューのいずれかのURLを指定できます。FIFO キューを指定する場合、定期的なタスクはサポートされないことに注意してください。	自動的に生成される	値を指定しない場合、Elastic Beanstalk は自動的にキューを作成します。
HttpPath	HTTP POST メッセージの送信先となるアプリケーションの相対パス。	/	

MimeType	HTTP POST リクエストで送信されるメッセージの MIME タイプ。	`application/json`	`application/json`
			
			`application/x-www-form-urlencoded`
			
			`application/xml`
			
			`text/plain`

名前	説明	デフォルト	有効な値
			カスタム MIME タイプ。
HttpConnections	Amazon EC2 インスタンス内にあるアプリケーションに同時に接続できる最大数。 **Note** Elastic Beanstalk コンソールを使用して環境を作成する場合は、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールによって、推奨値に上書きされます。	50	1 ~ 100
ConnectTimeout	アプリケーションへの接続が成功するまでの待機時間 (秒単位)	5	1 ~ 60
InactivityTimeout	アプリケーションへの既存の接続で、応答が返されるまで待機する時間 (秒単位) メッセージは、ワーカー環境枠でアプリケーションから 200 個 (OK) の応答をデーモンが受信するまで、または `RetentionPeriod` が有効期限切れになるまで再処理されます。	299	1 ~ 36000

名前	説明	デフォルト	有効な値
VisibilityTimeout	Amazon SQS キューからの着信メッセージが処理のためにロックされる時間の秒数。設定した時間数を経過すると、別のデーモンでの確認用に、メッセージがキューに再度表示されます。	300	`0 ~ 43200`
ErrorVisibilityTimeout	明示的なエラーで処理が失敗した後、Elastic Beanstalk が Amazon SQS キューにメッセージを返すまでに経過する時間 (秒単位)。	2 秒	`0 ~ 43200` 秒
Retention Period	メッセージが有効と判断され、アクティブな処理が行われるための時間 (秒単位)。	`345600`	`60 ~ 1209600`
MaxRetries	デッドレターキューにメッセージを移動する前に、Elastic Beanstalk がメッセージを処理するウェブアプリケーションに送信を試行する最大試行回数。	`10`	`1 ~ 100`

aws:elasticbeanstalk:trafficsplitting

環境に合わせてトラフィック分割デプロイを設定します。

この名前空間は、aws:elasticbeanstalk:command 名前空間の `DeploymentPolicy` オプションを `TrafficSplitting` に設定した場合に適用されます。デプロイのポリシーの詳細については、「the section called “デプロイオプション”」を参照してください。

名前空間: **`aws:elasticbeanstalk:trafficsplitting`**

名前	説明	デフォルト	有効な値
NewVersionPercent	Elastic Beanstalk が、デプロイする新しいアプリケーションバージョンを実行している環境インスタンスにシフトする受信クライアントトラフィックの初期の割合。	`10`	`1 ~ 100`
EvaluationTime	初回のデプロイが正常に完了してから、デプロイする新しいアプリケーションバージョンにすべての受信クライアントトラフィックをシフトするまで、Elastic Beanstalk が待機する時間 (分単位)。	`5`	`3 ~ 600`

aws:elasticbeanstalk:xray

デーモンを実行して、X-Ray AWS X-Ray 統合アプリケーションからトレース情報を中継します。
???

名前空間: **`aws:elasticbeanstalk:xray`**

名前	説明	デフォルト	有効な値
`XRayEnabled`	環境のインスタンスで X-Ray デーモンを実行するには、`true` に設定します。	`false`	`true` `false`

aws:elb:healthcheck

Classic Load Balancer のヘルスチェックを設定します。

名前空間: **`aws:elb:healthcheck`**

名前	説明	デフォルト	有効な値
HealthyThreshold	Elastic Load Balancing がインスタンスのヘルスステータスを変更するまでの間に、連続して成功したリクエスト数。	3	2 ~ 10
Interval	Elastic Load Balancing が、アプリケーションの Amazon EC2 インスタンスに対し、ヘルスチェックを実行する間隔。	10	5 ~ 300
Timeout	Elastic Load Balancing が、インスタンスを応答不能と判断するまで、応答を待機する時間 (秒単位)。	5	2 ~ 60
Unhealthy Threshold	Elastic Load Balancing がインスタンスのヘルスステータスを変更するまでの間に、連続して失敗したリクエスト数。	5	2 ~ 10
(廃止) Target	バックエンドインスタンス上での、ヘルスチェックの送信先。代わりに `Application Healthcheck URL` 名前空間の `aws:elasticbeanstalk:application` を使用します。	`TCP:80`	*`PROTOCOL`* の形式のターゲット: *`PORT/PATH`*

aws:elb:loadbalancer

環境の Classic Load Balancer を設定します。

この名前空間のオプションのいくつかは、aws:elb:listener名前空間のリスナー固有オプションを優先してサポートされなくなりました。サポート終了したこれらのオプションでは、標準ポートで 2 つの (1 つはセキュアで、もう 1 つはセキュアでない) リスナーのみを構成できます。

名前空間: `aws:elb:loadbalancer`

名前	説明	デフォルト	有効な値

CrossZone	ロードバランサーが各ゾーン内のみではなく、すべてのアベイラビリティーゾーン内のすべてのインスタンス間でトラフィックを均等にルーティングするかどうかを設定します。
	
	Note
	Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。
		`false`	`true`
			
			`false`

SecurityGroups	ロードバランサーに、作成した 1 つ以上のセキュリティグループを割り当てます。 `DisableDefaultEC2SecurityGroup` (aws:autoscaling:launchconfiguration) が に設定されている場合は必須です`true`。デフォルトの Elastic Beanstalk EC2 セキュリティグループをオプトアウトした負荷分散環境では、このオプションを 1 つ以上のセキュリティグループに提供する必要があります。詳細については、「EC2 セキュリティグループの管理」を参照してください。	なし	1 つまたは複数のセキュリティグループ ID。
ManagedSecurityGroup	新しいセキュリティグループを作成する代わりに、ご使用の環境のロードバランサーに既存のセキュリティグループを割り当	なし	セキュリティグループ ID。

名前	説明	デフォルト	有効な値
	てます。この設定を使用するには、この名前空間内の `SecurityGroups` 設定を変更し、セキュリティグループの IDを含めます。また、自動的に作成されたセキュリティグループがある場合は、その ID を削除します。 ご使用の環境の EC2 インスタンスに対する、ロードバランサーからのトラフィックを許可するために、Elastic Beanstalk は、マネージドセキュリティグループからの着信トラフィックを許可するルールを、インスタンスのセキュリティグループに追加します。		

(廃止) LoadBalancerHTTPPort	セキュアでないリスナーをリッスンするポート。	`80`	`OFF`
			
			`80`

(廃止) LoadBalancerPortProtocol	セキュアでないリスナーで使用するプロトコル。	`HTTP`	`HTTP`
			
			`TCP`

(廃止) LoadBalancerHTTPSPort	セキュアなリスナーをリッスンするポート。	`OFF`	`OFF`
			
			`443`
			
			`8443`

(廃止) LoadBalancerSSLPortProtocol	セキュアなリスナーで使用するプロトコル。	`HTTPS`	`HTTPS`
			
			`SSL`

(廃止) SSLCertificateId	セキュアなリスナーにバインドする SSL 証明書の Amazon リソースネーム (ARN)。	なし	

aws:elb:listener

Classic Load Balancer でデフォルトリスナー (ポート 80) を設定します。

名前空間: **`aws:elb:listener`**

名前	説明	デフォルト	有効な値
ListenerProtocol	リスナーによって使用されるプロトコルです。	HTTP	HTTP TCP
InstancePort	リスナーが EC2 インスタンスとの通信に使用するポートです。	80	1 ~ 65535
InstanceProtocol	リスナーが EC2 インスタンスとの通信に使用するプロトコルです。 これは ListenerProtocol と同じインターネットプロトコルレイヤーである必要があります。また、このリスナーとして同じ InstancePort を使用している他のリスナーと同じセキュリティレベルである必要があります。 たとえば、ListenerProtocol が HTTPS (安全な接続を使用したアプリケーションレイヤー) の場合、InstanceProtocol を HTTP (これもアプリケーションレイヤーですが、安全ではない接続を使用します) に設定できます。加えて、InstancePort を 80 に設定する場合、InstanceProtocol が HTTP に設定されているすべてのリスナーで InstancePort を 80 に設定する必要があります。	HTTP が ListenerProtocol の場合、HTTP TCP が ListenerProtocol の場合、TCP	HTTP または HTTPS。ListenerProtocol が HTTP または HTTPS のとき TCP または SSL。ListenerProtocol が TCP または SSL のとき
PolicyNames	このリスナーのポートに適用されるポリシー名のカンマ区切りリストです。この代わりに、aws:elb:policies 名前空間の	なし	

名前	説明	デフォルト	有効な値
	LoadBalancerPorts オプションの使用を推奨します。		

ListenerEnabled	このリスナーが有効かどうかを指定します。`false` を指定している場合、リスナーはロードバランサーに含まれません。	`true`	`true`
			
			`false`

aws:elb:listener:listener_port

Classic Load Balancer の追加リスナーを設定します。

名前空間: **`aws:elb:listener:`*`listener_port`***

名前	説明	デフォルト	有効な値

ListenerProtocol	リスナーによって使用されるプロトコルです。	`HTTP`	`HTTP`
			`HTTPS` `TCP`
			`SSL`

InstancePort	リスナーが EC2 インスタンスとの通信に使用するポートです。	*`listener_port`* と同じです。	1 ~ 65535

InstanceProtocol	リスナーが EC2 インスタンスとの通信に使用するプロトコルです。
	
	これは `ListenerProtocol` と同じインターネットプロトコルレイヤーである必要があります。また、このリスナーとして同じ `InstancePort` を使用している他のリスナーと同じセキュリティレベルである必要があります。
		`HTTP`。`List`
		`rotocol`
		が `HTTP`
		または
		`HTTPS` の
		とき
		
		`TCP`。`Liste`
		`rotocol`
		が `TCP` ま
			`HTTP`
			または
			`HTTPS`。`ListenerP`
			`rotocol`
			が `HTTP` ま
			または `HTTPS`
			のとき
			
			`TCP` または
			`SSL`。`ListenerP`
			`rotocol`

名前	説明	デフォルト	有効な値
	たとえば、`ListenerProtocol` が HTTPS (安全な接続を使用したアプリケーションレイヤー) の場合、`InstanceProtocol` を HTTP (これもアプリケーションレイヤーですが、安全ではない接続を使用します) に設定できます。加えて、`InstancePort` を `80` に設定する場合、`InstanceProtocol` が HTTP に設定されているすべてのリスナーで `InstancePort` を `80` に設定する必要があります。	たは SSL のとき	が TCP または SSL のとき
PolicyNames	このリスナーのポートに適用されるポリシー名のカンマ区切りリストです。変わりに名前空間 aws:elb:policies の [LoadBalancerPorts] オプションを使用することを推奨します。	なし	
SSLCertificateId	リスナーにバインドする SSL 証明書の Amazon リソースネーム (ARN)。	なし	
ListenerEnabled	このリスナーが有効かどうかを指定します。`false` を指定している場合、リスナーはロードバランサーに含まれません。	他のオプションが設定されている場合は `true` です。それ以外の場合は `false`。	`true` `false`

aws:elb:policies

Classic Load Balancer のデフォルトの維持およびグローバルロードバランサーポリシーを変更します。

名前空間: `aws:elb:policies`

名前	説明	デフォルト	有効な値

ConnectionDrainingEnabled	ロードバランサーが、処理中のリクエストを完了するために、異常なインスタンスや登録解除されたインスタンスへの既存の接続を維持するかどうかを指定します。
	
	Note
	Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールや EB CLI によって、推奨値に上書きされます。
		`false`	`true`
			
			`false`

ConnectionDrainingTimeout	ロードバランサーが Connection Draining の実行中にインスタンスへの接続を維持する最大秒数です。この時間を超えると強制的に接続を閉じます。 **Note** Elastic Beanstalk コンソールを使用して環境を作成する場合は、設定ファイルでこのオプションを設定することはできません。このオプションは、このコンソールによって、推奨値に上書きされます。	`20`	`1 ~ 3600`
ConnectionSettingIdleTimeout	接続を介してデータが送信または受信されるまで、ロードバランサーが待機する時	`60`	`1 ~ 3600`

名前	説明	デフォルト	有効な値
	間 (秒単位)。この時間が経過してもデータが送受信されなかった場合、ロードバランサーは接続を閉じます。		
LoadBalancerPorts	デフォルトポリシー (`AWSEB-ELB-StickinessPolicy`) が適用されるリスナーポートのカンマ区切りリストです。	なし	`:all`を使用してすべてのリスナーポートを指定できます。
Stickiness Cookie Expiration	各 Cookie の有効時間 (秒単位) です。デフォルトポリシー (`AWSEB-ELB-StickinessPolicy`) を使用します。	`0`	`0`～`1000000`
Stickiness Policy	セッション中にユーザーから受信するすべてのリクエストが、同じサーバーインスタンスに送信されるように、ユーザーのセッションを特定のサーバーインスタンスにバインドします。デフォルトポリシー (`AWSEB-ELB-StickinessPolicy`) を使用します。	`false`	`true` `false`

aws:elb:policies:policy_name

Classic Load Balancer の追加のロードバランーサポリシーを作成します。

名前空間: **`aws:elb:policies:`*`policy_name`***

名前	説明	デフォルト	有効な値
CookieName	アプリケーションによって生成され、`AppCookieStickinessPolicyType` ポリシーのセッション存続期間を管理する Cookie の名前です。こ	なし	

名前	説明	デフォルト	有効な値
	のポリシーは、HTTP/HTTPS リスナーにのみ関連付けることができます。		
InstancePorts	このポリシーが適用されるインスタンスポートカンマ区切りリストです。	なし	ポートのリスト、または `:all`
LoadBalancerPorts	このポリシーが適用されるリスナーのカンマ区切りリストです。	なし	ポートのリスト、または `:all`
ProxyProtocol	`ProxyProtocolPolicyType` ポリシーで、IP アドレスと TCP メッセージのリクエスト送信元のポートを含めるかどうかを指定します。このポリシーは、TCP/SSL リスナーにのみ関連付けることができます。	なし	`true false`
PublicKey	バックエンドサーバーを認証するときに使用する `PublicKeyPolicyType` ポリシーのパブリックキーの内容です。このポリシーは、バックエンドサーバーまたはリスナーに直接適用することはできません。`BackendServerAuthenticationPolicyType` ポリシーの中に含める必要があります。	なし	
PublicKeyPolicyNames	バックエンドサーバーへの認証を管理する `BackendServerAuthenticationPolicyType` ポリシーのポリシー名 (`PublicKeyPolicyType` ポリシーから) のカンマ区切りリストです。このポリシーは、HTTPS/SSL を使用しているバックエンドサーバーにのみ関連付けることができます。	なし	

名前	説明	デフォルト	有効な値
SSLProtocols	ロードバランサーで許可される暗号とプロトコルを定義するための、`SSLNegotiationPolicyType` ポリシーで有効化される、SSL プロトコルのカンマ区切りリスト。このポリシーは、HTTPS/SSL リスナーにのみ関連付けることができます。	なし	
SSLReferencePolicy	セキュリティのベストプラクティスに準拠し、ロードバランサーが受け入れる暗号とプロトコルを定義するポリシーに対してアクティブ化する事前定義された AWS セキュリティ`SSLNegotiationPolicyType` ポリシーの名前。このポリシーは、HTTPS/SSL リスナーにのみ関連付けることができます。	なし	
Stickiness Cookie Expiration	各 Cookie の有効時間（秒単位）です。	`0`	`0 ~ 1000000`
Stickiness Policy	セッション中にユーザーから受信するすべてのリクエストが、同じサーバーインスタンスに送信されるように、ユーザーのセッションを特定のサーバーインスタンスにバインドします。	`false`	`true false`

aws:elbv2:listener:default

Application Load Balancer または Network Load Balancer のデフォルトのリスナー (ポート 80) を設定します。

この名前空間は、共有ロードバランサーを使用する環境には適用されません。共有ロードバランサーにはデフォルトのリスナーがありません。

名前空間: `aws:elbv2:listener:default`

名前	説明	デフォルト	有効な値
DefaultProcess	ルールが適合しないときにトラフィックを転送するプロセスの名前。	`default`	プロセス名。

ListenerEnabled	リスナーを無効にするには、`false` に設定します。ポート 80 でデフォルトリスナーを無効にするには、このオプションを使用できます。	`true`	`true`
			
			`false`

Protocol	処理するトラフィックのプロトコル。	アプリケーションロードバランサー: `HTTP` network load balancer: `TCP`	アプリケーションロードバランサー: `HTTP`、`HTTPS` network load balancer: `TCP`
Rules	リスナーに適用するルールのリスト このオプションは、Application Load Balancer を使用する環境にのみ適用されます。	なし	ルール名のリスト (コンマ区切り)。
SSLCertificateArns	リスナーにバインドする SSL 証明書の Amazon リソースネーム (ARN)。	なし	IAM または ACM に保存された証明書の ARN。

名前	説明	デフォルト	有効な値
	このオプションは、Application Load Balancer を使用する環境にのみ適用されます。		
SSLPolicy	リスナーに適用するセキュリティポリシーを指定します。 このオプションは、Application Load Balancer を使用する環境にのみ適用されます。	なし (ELB のデフォルト)	ロードバランサーセキュリティポリシーの名前。

aws:elbv2:listener:listener_port

Application Load Balancer または Network Load Balancer の追加リスナーを設定します。

> **Note**
>
> 共有 Application Load Balancer の場合は、`Rule` オプションのみを指定できます。その他のオプションは、共有ロードバランサーには適用されません。

名前空間: **`aws:elbv2:listener:`*`listener_port`***

名前	説明	デフォルト	有効な値
DefaultProcess	ルールが一致しないときにトラフィックを転送するプロセスの名前。	`default`	プロセス名。
ListenerEnabled	リスナーを無効にするには、`false` に設	`true`	`true`

名前	説明	デフォルト	有効な値
	定します。ポート 80 でデフォルトリスナーを無効にするには、このオプションを使用できます。		`false`
Protocol	処理するトラフィックのプロトコル。	アプリケーションロードバランサー: `HTTP` network load balancer: `TCP`	アプリケーションロードバランサー: `HTTP`、`HTTPS` network load balancer: `TCP`
Rules	リスナーに適用するルールのリスト このオプションは、Application Load Balancer を使用する環境にのみ適用されます。 環境が共有 Application Load Balancer を使用し、このオプションをリスナーに指定しない場合、Elastic Beanstalk は自動的に `default` ルールをポート 80 リスナーに関連付けます。	なし	ルール名のリスト (コンマ区切り)。

名前	説明	デフォルト	有効な値
SSLCertificateArns	リスナーにバインドする SSL 証明書の Amazon リソースネーム (ARN)。 このオプションは、Application Load Balancer を使用する環境にのみ適用されます。	なし	IAM または ACM に保存された証明書の ARN。
SSLPolicy	リスナーに適用するセキュリティポリシーを指定します。 このオプションは、Application Load Balancer を使用する環境にのみ適用されます。	なし (ELB のデフォルト)	ロードバランサーセキュリティポリシーの名前。

aws:elbv2:listenerrule:rule_name

Application Load Balancer 用のリスナールールを定義します。リクエストがルールのホスト名あるいはパスと一致する場合、ロードバランサーはこのリクエストを指定されたプロセスに転送します。ルールを使用するには、`Rules` 名前空間の `aws:elbv2:listener:`*`listener_port`* オプションでそのルールをリスナーに追加します。

> **Note**
>
> この名前空間は、network load balancerを使用する環境には適用されません。

名前空間: **`aws:elbv2:listenerrule:`*`rule_name`***

名前	説明	デフォルト	有効な値
HostHeaders	一致させるホスト名のリスト。例えば、`my.example.com` 。	専用ロードバランサー: なし 共有ロードバランサー: 環境の CNAME	それぞれの名前の長さは最大 128 文字です。パターンには、大文字および小文字、数字、ハイフン (–)、および最大 3 つのワイルドカード文字 (* は 0 文字以上に、? はちょうど 1 文字にマッチ) を含めることができます。カンマで区切って、複数の名前をリストできます。Application Load Balancer は、最大 5 つの `HostHeader` ルールと `PathPattern` ルールの組み合わせをサポートします。 詳細については、Application Load Balancer のユーザーガイドの「ホストの条件」を参照してください。
PathPatterns	一致させるパスパターン (例えば、`/img/*`)。	なし	各パターンの長さは最大 128 文字です。パターンには、大文字および小文字、数字、ハイフン (–)、お

名前	説明	デフォルト	有効な値
	このオプションは、アプリケーションロードバランサーを使用する環境にのみ適用されます。		および最大 3 つのワイルドカード文字 (* は 0 文字以上に、? はちょうど1 文字にマッチ) を含めることができます。複数のパスパターンをカンマで区切って追加できます。Application Load Balancer は、最大 5 つの `HostHeader` ルールと `PathPattern` ルールの組み合わせをサポートします。 詳細については、Application Load Balancer ユーザーガイドの「パスの条件」を参照してください。
Priority	複数のルールが一致する場合の、このルールの優先順位。低い番号が優先されます。2 つのルールに同じ優先順位を設定することはできません。 共有ロードバランサーを使用すると、Elastic Beanstalk では、共有環境全体でルールの優先順位が相対的なものとして扱われ、作成時に絶対的な優先順位にマッピングされます。	`1`	`1 ~ 1000`
Process	このルールがリクエストに適合する場合にトラフィックを転送するプロセスの名前。	`default`	プロセス名。

aws:elbv2:loadbalancer

Application Load Balancer を設定します。

共有ロードバランサーの場合、`SharedLoadBalancer` および `SecurityGroups` オプションのみが有効です。

> **Note**
> この名前空間は、Network Load Balancer を使用する環境には適用されません。

名前空間: **`aws:elbv2:loadbalancer`**

名前	説明	デフォルト	有効な値
AccessLogsS3Bucket	アクセスログが保存される Amazon S3 バケット。このバケットは環境と同じリージョンに置かれ、ロードバランサーからの書き込みアクセスを許可している必要があります。	なし	バケット名。
AccessLogsS3Enabled	アクセスログストレージを有効にします。	`false`	`true` `false`
AccessLogsS3Prefix	アクセスログ名に前置するプレフィックス。デフォルトでは、ロードバランサーがログをアップロードする先は、指定したバケットの AWSLogs というディレクトリです。別の場所に AWSLogs ディレクトリを配置する場合は、プレフィックスを指定します。	なし	
IdleTimeout	クライアントとインスタンスへの接続を閉じる前に、リクエストの完了を待機する時間 (秒単位)。	なし	`1 ~ 3600`

名前	説明	デフォルト	有効な値
ManagedSecurityGroup	新しいセキュリティグループを作成する代わりに、ご使用の環境のロードバランサーに既存のセキュリティグループを割り当てます。この設定を使用するには、この名前空間の `SecurityGroups` 設定を更新してセキュリティグループの ID を含め、自動的に作成されたセキュリティグループの ID がある場合はそれを削除します。 ロードバランサーから環境の EC2 インスタンスへのトラフィックを許可するために、Elastic Beanstalk は、マネージドセキュリティグループからの着信トラフィックを許可するルールを、インスタンスのセキュリティグループに追加します。	ロードバランサー用に Elastic Beanstalk が作成するセキュリティグループ。	セキュリティグループ ID。

名前	説明	デフォルト	有効な値
SecurityGroups	ロードバランサーにアタッチするセキュリティグループのリスト。 `DisableDefaultEC2SecurityGroup` (aws:autoscaling:launchconfiguration) が に設定されている場合は必須です`true`。デフォルトの Elastic Beanstalk EC2 セキュリティグループをオプトアウトした負荷分散環境では、このオプションを 1 つ以上のセキュリティグループに提供する必要があります。詳細については、「EC2 セキュリティグループの管理」を参照してください。 共有ロードバランサーで、この値を指定しない場合、Elastic Beanstalk は、管理する既存のセキュリティグループがロードバランサーに既にアタッチされているかどうかを確認します。ロードバランサーにアタッチされたセキュリティグループがない場合、Elastic Beanstalk はそれを作成しロードバランサーにアタッチします。Elastic Beanstalk は、ロードバランサーを共有する最後の環境が終了すると、このセキュリティグループを削除します。 ロードバランサーのセキュリティグループは、Amazon EC2 インスタンスのセキュリティグループの入力	ロードバランサー用に Elastic Beanstalk が作成するセキュリティグループ。	セキュリティグループ ID のカンマ区切りのリスト。

名前	説明	デフォルト	有効な値
	ルールを設定するために使用されます。		

名前	説明	デフォルト	有効な値
SharedLoadBalancer	共有ロードバランサーの Amazon リソースネーム (ARN)。このオプションは、Application Load Balancer にのみ関連します。aws:elasticbeanstalk:environment 名前空間の `LoadBalancerIsShared` オプションに `true` が設定されている場合に必須です。環境の作成後に共有ロードバランサーの ARN を変更することはできません。 有効な値の基準: • 環境が配置されている AWS リージョンで有効でアクティブなロードバランサーである必要があります。 • 環境と同じ Amazon Virtual Private Cloud (Amazon VPC) に存在する必要があります。 • 別の環境の専用ロードバランサーとして Elastic Beanstalk によって作成されたロードバランサーにはできません。これらの専用ロードバランサーは、プレフィックス `awseb-` で識別できます。 例: `arn:aws:elasticloadbalancing:us-east-2:123456789012:lo`	なし	ここに記述されたすべての基準を満たす有効なロードバランサーの ARN。

名前	説明	デフォルト	有効な値
	`adbalancer/app/FrontEndLB/0dbf78d8ad96abbc`		

aws:rds:dbinstance

アタッチされた Amazon RDS DB インスタンスを設定します。

名前空間: **`aws:rds:dbinstance`**

名前	説明	デフォルト	有効な値
DBAllocatedStorage	割り当て済みデータベースストレージのサイズ（ギガバイト単位）。	MySQL: 5 Oracle: 10 sqlserver-se: 200 sqlserver-ex: 30 sqlserver-web: 30	MySQL: 5-1024 Oracle: 10-1024 sqlserver: 変更することはできません

DBDeletionPolicy	環境が終了された場合に、DB インスタンスを保持するか、削除するか、そのスナップショットを作成するかを指定します。
	
	このオプションは、`HasCoupledDatabase` と組み合わせて使用します。この名前空間のオプションでもあります。
		`Delete`	`Delete`
			
			`Retain`
			
			`Snapshot`

名前	説明	デフォルト	有効な値
	Warning DB インスタンスを削除した場合、そのデータは復元できません。		

DBEngine	このインスタンスに使用するデータベースエンジンの名前。	`mysql`	`mysql`
			`oracle-se1`
			`sqlserver-ex`
			`sqlserver-web`
			`sqlserver-se`
			`postgres`

DBEngineVersion	データベースエンジンのバージョン番号。	`5.5`	
DBInstanceClass	データベースインスタンスのタイプ。	`db.t2.micro` (Amazon VPC を実行しない環境では `db.m1.large`)	詳細については、Amazon Relational Database Service ユーザーガイドの「DB インスタンスクラス」を参照してください。
DBPassword	データベースインスタンスのマスターユーザーパスワード。	なし	

名前	説明	デフォルト	有効な値
DBSnapshotIdentifier	リストア元の DB スナップショットの識別子。	なし	
DBUser	DB インスタンスのマスターユーザーの名前。	ebroot	

HasCoupledDatabase	DB インスタンスが環境にカップリングされているかどうかを指定します。`true` にした場合、Elastic Beanstalk によって環境にカップリングされた新しい DB インスタンスが作成されます。`false` にした場合は、Elastic Beanstalk は環境から DB インスタンスのデカップリングを開始します。
	
	このオプションは、`DBDeletionPolicy` と組み合わせて使用します。この名前空間のオプションでもあります。
	
	Note
	注: 以前のデータベースをデカップリングした後、この値を `true` に戻した場合、Elastic Beanstalk は以前のデータベースオプション設定を使用して新しいデータベースを作成します。ただし、環境のセキュリティを維持するために、既存の `DBUser` および `DBPassword` 設定は保持されません。`DBUser` および `DBPassword` を再度指定する必要があります。
		`false`	`true`
			
			`false`

名前	説明	デフォルト	有効な値

MultiAZDatabase	データベースインスタンス Multi-AZ デプロイを作成する必要があるかどうかを指定します。Amazon Relational Database Service (RDS) のマルチ AZ 配置の詳細については、Amazon Relational Database Service ユーザーガイドの、「リージョン、アベイラビリティーゾーン、およびローカルゾーン」を参照してください。	`false`	`true`
			
			`false`

プラットフォーム固有のオプション

一部の Elastic Beanstalk プラットフォームは、プラットフォームに固有のオプション名前空間を定義します。これらの名前空間とそのオプションは、プラットフォームごとに以下にリストされています。

> **Note**
>
> 以前は、Amazon Linux AMI (Amazon Linux 2 以前) に基づくプラットフォームバージョンでは、次の 2 つの機能とそれぞれの名前空間はプラットフォーム固有の機能と見なされ、プラットフォームごとにここにリストされていました。
>
> - 静的ファイルのプロキシ設定 – `aws:elasticbeanstalk:environment:proxy:staticfiles`
> - AWS X-Ray サポート – `aws:elasticbeanstalk:xray`
>
> Amazon Linux 2 プラットフォームバージョンでは、Elastic Beanstalk は、サポートするすべてのプラットフォームで一貫した方法でこれらの機能を実装します。関連する名前空間が「the section called “汎用オプション”」ページにリストされるようになりました。名前空間の名前が異なるプラットフォームについては、このページでのみ言及していました。

プラットフォーム

- Docker プラットフォームのオプション
- Go プラットフォームのオプション

- Java SE プラットフォームのオプション
- Java と Tomcat プラットフォームのオプション
- Linux プラットフォームオプション上の .NET Core
- .NET プラットフォームのオプション
- Node.js プラットフォームのオプション
- PHP プラットフォームのオプション
- Python プラットフォームのオプション
- Ruby プラットフォームのオプション

Docker プラットフォームのオプション

以下に示す Docker 固有の設定オプションは、Docker および Preconfigured Docker プラットフォームに使用します。

> **Note**
>
> これらの設定オプションは以下に適用されません。
>
> - Docker Compose を使用した Docker プラットフォーム (Amazon Linux 2)
> - マルチコンテナ Docker プラットフォーム (Amazon Linux AMI AL1) - このプラットフォームは廃止されました

名前空間: **`aws:elasticbeanstalk:environment:proxy`**

名前	説明	デフォルト	有効な値
ProxyServer	プロキシとして使用するウェブサーバーを指定します。	`nginx`	`nginx` none – Amazon Linux AMおよびDocker w/DCのみ

Go プラットフォームのオプション

Amazon Linux AMI (Amazon Linux 2 以前) プラットフォームオプション

名前空間: **`aws:elasticbeanstalk:container:golang:staticfiles`**

静的ファイルを処理するようにプロキシサーバーを設定するには、次の名前空間を使用できます。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接処理します。これにより、アプリケーションで処理する必要があるリクエストの数が減ります。

プロキシサーバーで処理されるパスを、静的アセットを含むソースコード内のフォルダにマッピングします。この名前空間で定義される各オプションは、それぞれ異なるパスをマッピングします。

名前	値
プロキシサーバーでファイルが配信されるパス。 例: `/images` のファイルを配信する *`subdomain`* `.eleasticbeanstalk.com/images`	ファイルを含むフォルダの名前。 例: ソースバンドルの最上位にある `staticimages` という名前のフォルダのファイルを配信する `staticimages` 。

Java SE プラットフォームのオプション

Amazon Linux AMI (Amazon Linux 2 以前) プラットフォームオプション

名前空間: **`aws:elasticbeanstalk:container:java:staticfiles`**

静的ファイルを処理するようにプロキシサーバーを設定するには、次の名前空間を使用できます。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接処理します。これにより、アプリケーションで処理する必要があるリクエストの数が減ります。

プロキシサーバーで処理されるパスを、静的アセットを含むソースコード内のフォルダにマッピングします。この名前空間で定義される各オプションは、それぞれ異なるパスをマッピングします。

名前	値
プロキシサーバーでファイルが配信されるパス。 例: `/images` のファイルを配信する *`subdomain`* `.eleasticbeanstalk.com/images`	ファイルを含むフォルダの名前。 例: ソースバンドルの最上位にある `staticimages` という名前のフォルダのファイルを配信する `staticimages` 。

Java と Tomcat プラットフォームのオプション

名前空間: **`aws:elasticbeanstalk:application:environment`**

名前	説明	デフォルト	有効な値
JDBC_CONNECTION_STRING	外部データベースへの接続文字列です。	該当なし	該当なし

詳細については「環境変数およびその他のソフトウェア設定」を参照してください。

名前空間: **`aws:elasticbeanstalk:container:tomcat:jvmoptions`**

名前	説明	デフォルト	有効な値
JVM Options	起動時にコマンドラインオプションを JVM に渡します。	該当なし	該当なし
Xmx	JVM の最大ヒープサイズ。	`256m`	該当なし
XX:MaxPermSize	クラス定義と関連するメタデータの保存に使用される JVM ヒープのセクション。 **Note** このオプションは Java 8 以前の Java バージョンにのみ適用され、Amazon Linux 2 以降に基づく Elastic	`64m`	該当なし

名前	説明	デフォルト	有効な値
	Beanstalk Tomcat プラットフォームではサポートされません。		
Xms	JVM の初期ヒープサイズ。	`256m`	該当なし
`optionName`	Tomcat プラットフォームで定義されているオプションに加えて、任意の JVM オプションを指定します。	該当なし	該当なし

名前空間: **`aws:elasticbeanstalk:environment:proxy`**

名前	説明	デフォルト	有効な値

GzipCompression	レスポンスの圧縮を無効にするには、`false` に設定します。
	
	Amazon Linux AMI (Amazon Linux 2 以前) プラットフォームバージョンでのみ有効です。
		`true`	`true`
			
			`false`

ProxyServer	環境のインスタンスで使用するように、プロキシを設定します。このオプションを`apache`に設定した場合、Elastic Beanstalk は Apache 2.4 を使用します。
	
	プロキシの設定に互換性がないため、アプリケーションで Apache 2.2 からの移行の準備が整っていない場合は、`apache/2.2` に設定します。この値は、Amazon Linux AMI (Amazon Linux 2 以前) プラットフォームバージョンでのみ有効です。
	
	nginx を使用するには、`nginx` に設定します。これは、Amazon Linux 2 プラット
		`nginx` (Amazon Linux 2)
		
		`apache` (Amazon Linux AMI)
			`apache`
			
			`apache/2.2` – Amazon Linux AMI のみ
			
			`nginx`

名前	説明	デフォルト	有効な値
	フォームバージョン以降のデフォルトです。 詳細については、「プロキシサーバーを設定します」を参照してください。		

Linux プラットフォームオプション上の .NET Core

名前空間: **`aws:elasticbeanstalk:environment:proxy`**

名前	説明	デフォルト	有効な値

ProxyServer	プロキシとして使用するウェブサーバーを指定します。	`nginx`	`nginx`
			
			`none`

.NET プラットフォームのオプション

名前空間: **`aws:elasticbeanstalk:container:dotnet:apppool`**

名前	説明	デフォルト	有効な値

Target Runtime	アプリケーションの .NET Framework のバージョンを選択します。	`4.0`	`2.0`
			
			`4.0`

Enable 32-bit Applications	32 ビットアプリケーションを実行するには `True` に設定します。	`False`	`True`
			
			`False`

Node.js プラットフォームのオプション

名前空間: **`aws:elasticbeanstalk:environment:proxy`**

名前	説明	デフォルト	有効な値

ProxyServer	環境のインスタンスで使用するように、プロキシを設定します。	`nginx`	`apache`
			
			`nginx`

Amazon Linux AMI (Amazon Linux 2 以前) プラットフォームオプション

名前空間: **`aws:elasticbeanstalk:container:nodejs`**

名前	説明	デフォルト	有効な値
NodeCommand	Node.js アプリケーションの起動に使用するコマンド。空の文字列を指定すると、`app.js`、`server.js` 、`npm start` の順で使用されます。	""	該当なし
NodeVersion	Node.js のバージョン。例えば、`4.4.6` サポートされる Node.js のバージョンは、Node.js プラットフォームのバージョンによって異なります。現在サポートされているバージョンの一覧については、AWS Elastic Beanstalk プラットフォームのドキュメントにある Node.js を参照してください。 **Note** 使用しているバージョンの Node.js に対するサポートがプラットフォームバージョンから削除された場合は、プラットフォームの更新に	varies	varies

名前	説明	デフォルト	有効な値
	先立って、バージョン設定を変更または削除する必要があります。これは、1 つ以上のバージョンの Node.js でセキュリティの脆弱性が検出された場合に発生することがあります この場合、設定した NodeVersion をサポートしない新しいバージョンのプラットフォームにアップグレードしようとすると、失敗します。新しい環境の作成を回避するには、古いプラットフォームバージョンと新しいプラットフォームバージョンの両方でサポートされている Node.js バージョンに NodeVersion 設定オプションを変更するか、オプション設定を削除してから、プラットフォームの更新を実行します。		
GzipCompression	gzip 圧縮が有効かどうかを指定します。ProxyServer を none に設定すると、gzip の圧縮は無効になります。	`false`	`true` `false`

ProxyServer	Node.js へのプロキシ接続に使用するウェブサーバーを指定します。ProxyServer を none に設定すると、静的なファイルのマッピングが有効にならず、gzip の圧縮は無効になります。	`nginx`	`apache`
			`nginx`
			`none`

名前空間: **`aws:elasticbeanstalk:container:nodejs:staticfiles`**

静的ファイルを処理するようにプロキシサーバーを設定するには、次の名前空間を使用できます。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーション

にリクエストをルーティングする代わりにファイルを直接処理します。これにより、アプリケーションで処理する必要があるリクエストの数が減ります。

プロキシサーバーで処理されるパスを、静的アセットを含むソースコード内のフォルダにマッピングします。この名前空間で定義される各オプションは、それぞれ異なるパスをマッピングします。

> **Note**
>
> `aws:elasticbeanstalk:container:nodejs::ProxyFiles` が `none` に設定されている場合、静的ファイル設定は適用されません。

名前	値
プロキシサーバーでファイルが配信されるパス。 例: `/images` のファイルを配信する *`subdomain`* `.elasticbeanstalk.com/images`	ファイルを含むフォルダの名前。 例: ソースバンドルの最上位にある `staticimages` という名前のフォルダのファイルを配信する `staticimages` 。

PHP プラットフォームのオプション

名前空間: **`aws:elasticbeanstalk:container:php:phpini`**

名前	説明	デフォルト	有効な値
document_root	パブリック側のウェブルートとして扱うプロジェクトの子ディレクトリを指定します。	`/`	空の文字列は「`/`」として扱われます。または、「`/`」から始まる文字列を指定します。
memory_limit	PHP 環境に割り当てるメモリサイズ。	`256M`	該当なし

zlib.output_compression	PHP の出力に圧縮を使用するかどうかを指定します。	`Off`	`On`
			
			`Off`

名前	説明	デフォルト	有効な値
			`true` `false`

allow_url_fopen	ウェブサイトや FTP サーバーなど、リモートの場所からデータを取得する PHP のファイル機能を許可するかどうかを指定します。	`On`	`On`
			`Off`
			`true`
			`false`

display_errors	エラーメッセージを出力に含めるかどうかを指定します。	`Off`	`On` `Off`
max_execution_time	環境によって終了される前に、スクリプトを実行できる最大時間（秒）を設定します。	`60`	`0`～`9223372036854775807`（PHP_INT_MAX）
composer_options	composer.phar install コマンドで、Composer を使用して依存関係をインストールするときに使用するカスタムオプションを設定します。詳細については、getcomposer.org ウェブサイトの「インストール」を参照してください。	該当なし	該当なし

名前空間: **`aws:elasticbeanstalk:environment:proxy`**

名前	説明	デフォルト	有効な値
ProxyServer	環境のインスタンスで使用するように、プロキシを設定します。	`nginx`	`apache` `nginx`

> **Note**
>
> PHP プラットフォームの詳細については、Elastic Beanstalk PHP プラットフォームを使用する を参照してください。

Python プラットフォームのオプション

名前空間: **`aws:elasticbeanstalk:application:environment`**

名前	説明	デフォルト	有効な値
DJANGO_SETTINGS_MODULE	使用する設定ファイルを指定します。	該当なし	該当なし

詳細については「環境変数およびその他のソフトウェア設定」を参照してください。

名前空間: **`aws:elasticbeanstalk:container:python`**

名前	説明	デフォルト	有効な値
WSGIPath	WSGI アプリケーションを含むファイル。このファイルには呼び出し可能な `application` が必要です。	Amazon Linux 2 Python プラットフォームバージョン: `application` Amazon Linux AMI Python プラットフォームバージョン: `application.py`	該当なし

名前	説明	デフォルト	有効な値
NumProcesses	WSGI アプリケーションを実行するときに、プロセスグループで開始するデーモンプロセス数。	1	該当なし
NumThreads	WSGI アプリケーションを実行するときに、プロセスグループ内の各デーモンプロセスでリクエストを処理するために作成するスレッド数。	15	該当なし

名前空間: **`aws:elasticbeanstalk:environment:proxy`**

名前	説明	デフォルト	有効な値
ProxyServer	環境のインスタンスで使用するように、プロキシを設定します。	`nginx`	`apache` `nginx`

Amazon Linux AMI (Amazon Linux 2 以前) プラットフォームオプション

名前空間: **`aws:elasticbeanstalk:container:python:staticfiles`**

静的ファイルを処理するようにプロキシサーバーを設定するには、次の名前空間を使用できます。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接処理します。これにより、アプリケーションで処理する必要があるリクエストの数が減ります。

プロキシサーバーで処理されるパスを、静的アセットを含むソースコード内のフォルダにマッピングします。この名前空間で定義される各オプションは、それぞれ異なるパスをマッピングします。

デフォルトでは、Python 環境のプロキシサーバーは `static` パスにある `/static` という名前のフォルダにあるすべてのファイルを提供します。

名前空間: **`aws:elasticbeanstalk:container:python:staticfiles`**

名前	値
プロキシサーバーでファイルが配信されるパス。	ファイルを含むフォルダの名前。

名前	値
例: `/images` のファイルを配信する *subdomain* `.elasticbeanstalk.com/images`	例: ソースバンドルの最上位にある `staticimages` という名前のフォルダのファイルを配信する `staticimages` 。

Ruby プラットフォームのオプション

名前空間: **`aws:elasticbeanstalk:application:environment`**

名前	説明	デフォルト	有効な値
RAILS_SKIP_MIGRATIONS	ユーザーのアプリケーションの代わりに `` `rake db:migrate` `` を実行するか、スキップするかを指定します。この設定は Rails 3 アプリケーションにのみ適用されます。	`false`	`true` `false`
RAILS_SKIP_ASSET_COMPILATION	コンテナでユーザーのアプリケーションの代わりに `` `rake assets:precompile` `` を実行するか、スキップするかを指定します。この設定も Rails 3 アプリケーションにのみ適用されます。	`false`	`true` `false`
BUNDLE_WITHOUT	Gemfile から依存関係をインストールするときに無視するグループのコロン（「:」）区切りリスト。	`test:development`	該当なし
RACK_ENV	アプリケーションを実行できる環境ステージを指定します。開発、本番環境、テストが含まれた一般的な環境の例	`production`	該当なし

詳細については「環境変数およびその他のソフトウェア設定」を参照してください。

カスタム オプション

オプションおよび他の設定ファイルの aws:elasticbeanstalk:customoption ブロックで読み取りが可能な値を定義するために、Resources 名前空間を使用します。単一の設定ファイルでユーザー指定の設定を収集するために、カスタムオプションを使用します。

たとえば、環境を起動するユーザーにより設定される、リソースを定義する複雑な設定ファイルがあるとします。カスタムオプションの値を取得するために Fn::GetOptionSetting を使用する場合は、ユーザーにより簡単に検出でき、修正できる、別の設定ファイルのオプションの定義を指定できます。

また、設定オプションであるため、カスタムオプションは API レベルで設定ファイルの値を上書きするように設定できます。詳細については、「優先順位」を参照してください。

カスタムオプションは、その他のオプションと同様に定義されます。

```
option_settings:
  aws:elasticbeanstalk:customoption:
    option name: option value
```

たとえば、次の設定ファイルは ELBAlarmEmail という名前のオプションを作成し、someone@example.com に値を設定します。

```
option_settings:
  aws:elasticbeanstalk:customoption:
    ELBAlarmEmail: someone@example.com
```

他の場所で、設定ファイルは Fn::GetOptionSetting 属性の値を入力するために、Endpoint でオプションを読み取る SNS トピックを定義します。

```
Resources:
  MySNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint:
            Fn::GetOptionSetting:
              OptionName: ELBAlarmEmail
              DefaultValue: nobody@example.com
```

```
          Protocol: email
```

`Fn::GetOptionSetting` を使用したスニペットの他の例は、Elastic Beanstalk 環境リソースの追加とカスタマイズ にあります。

設定ファイル (`.ebextensions`) による高度な環境のカスタマイズ

ウェブアプリケーションのソースコードに設定 AWS Elastic Beanstalk ファイル (`.ebextensions`) を追加して環境を設定し、そこに含まれる AWS リソースをカスタマイズできます。設定ファイルは、ファイル拡張子を `.config` とする YAML または JSON 形式のドキュメントで、`.ebextensions` というフォルダ内に配置してアプリケーションソースバンドルにデプロイします。

Example .ebextensions/network-load-balancer.config

この例では、簡単な設定変更を行います。環境ロードバランサーのタイプを Network Load Balancer に設定する設定オプションを変更します。

```
option_settings:
  aws:elasticbeanstalk:environment:
    LoadBalancerType: network
```

YAML 形式の設定ファイルを使用することをお勧めします。JSON よりも読みやすいためです。YAML では、コメント、複数行のコマンド、さまざまな引用符の使い方などがサポートされています。ただし、YAML を使用しても JSON を使用しても Elastic Beanstalk 設定ファイルで設定の変更を行う方法は同じです。

> **ヒント**
>
> 新しい設定ファイルを開発またはテストする場合、デフォルトアプリケーションを実行するクリーンな環境を起動し、そこでデプロイします。設定ファイルが正しくフォーマットされていない場合、新しい環境の起動は失敗し、回復不可能になります。

設定ファイルの `option_settings` セクションは、設定オプションの値を定義します。設定オプションを使用すると、Elastic Beanstalk 環境、環境内の AWS リソース、およびアプリケーションを

実行するソフトウェアを設定できます。設定ファイルは、設定オプションを設定するいくつかの方法の 1 つに過ぎません。

このセクションでは`Resources`、アプリケーションの環境内のリソースをさらにカスタマイズし、設定オプションによって提供される機能以外の追加の AWS リソースを定義できます。Elastic Beanstalk が環境の作成に使用する AWS CloudFormation、 でサポートされている任意のリソースを追加および設定できます。

設定ファイルのその他のセクション（`packages`、`sources`、`files`、`users`、`groups`、`commands`、`container_commands`、`services`）では、環境内で起動される EC2 インスタンスを設定できます。サーバーが環境内で起動される場合、Elastic Beanstalk は常にこれらのセクションで定義されたオペレーションを実行し、アプリケーションのためのオペレーティングシステムとストレージシステムを準備します。

一般的に使用される .ebextensions の例については、「Elastic Beanstalk Configuration Files Repository」を参照してください。

要件

- 場所 – Elastic BCreate an example applicationeanstalk は、デプロイに存在するすべての `.ebextensions` フォルダを処理します。ただし、ソースバンドルのルートにある `.ebextensions` という単一のフォルダにすべての設定ファイルを配置することをお勧めします。ドットで開始するフォルダは、ファイルブラウザで非表示になっている場合があるため、ソースバンドル作成時にフォルダが追加されていることを確認します。詳細については、「Elastic Beanstalk アプリケーションソースバンドルを作成する」を参照してください。
- 名前付け - 設定ファイルには、`.config` ファイル拡張子を加える必要があります。
- 形式 - 設定ファイルは YAML または JSON 仕様に準拠する必要があります。

  YAML を使用する場合は、必ずスペースを使用して、異なるネストレベルごとにキーをインデントします。YAML の詳細については、YAML Ain't Markup Language (YAML™) Version 1.1 を参照してください。
- 一意性 - 各設定ファイルで各キーを 1 回のみ使用します。

> **警告**
>
> 同じ設定ファイルでキー (例: `option_settings`) を 2 回使用すると、いずれかのセクションは削除されます。重複セクションを 1 つのセクションにまとめるか、別々の設定ファイルに配置します。

デプロイするプロセスは、環境を管理するために使用しているクライアントによって少し異なります。詳細については、次のセクションを参照してください。

- Elastic Beanstalk コンソール
- EB CLI
- AWS CLI

トピック

- オプション設定
- Linux サーバーでのソフトウェアのカスタマイズ
- Windows サーバーでのソフトウェアのカスタマイズ
- Elastic Beanstalk 環境リソースの追加とカスタマイズ

オプション設定

`option_settings` キーを使用すると、Elastic Beanstalk 設定を変更し、環境変数を使用してアプリケーションから取得できる変数を定義できます。一部の名前空間では、パラメータの数を増やし、パラメータ名を指定できます。名前空間と設定オプションの一覧については、「設定オプション」を参照してください。

オプション設定は、環境の作成時または環境の更新時に直接環境に適用することもできます。環境に直接適用される設定は、設定ファイルの同じオプションの設定を上書きします。環境の設定から設定を削除すると、設定ファイルの設定が有効になります。詳細については、「優先順位」を参照してください。

構文

オプション設定の標準的な構文は、`namespace`、`option_name` 、`value` キーを格納したオブジェクトの配列です。

```
option_settings:
  - namespace:  namespace
    option_name:  option name
    value:  option value
  - namespace:  namespace
    option_name:  option name
    value:  option value
```

namespace キーはオプションです。名前空間を指定しない場合に使用されるデフォルトは aws:elasticbeanstalk:application:environment です。

```
option_settings:
  - option_name:  option name
    value:  option value
  - option_name:  option name
    value:  option value
```

Elastic Beanstalk では、オプション設定の簡易構文もサポートされており、名前空間内のキーと値のペアとしてオプションを指定できます。

```
option_settings:
  namespace:
    option name: option value
    option name: option value
```

例

以下の例では、Tomcat プラットフォーム固有のオプションを名前空間 aws:elasticbeanstalk:container:tomcat:jvmoptions と MYPARAMETER という環境プロパティに設定します。

標準の YAML 形式では、以下のようになります。

Example .ebextensions/options.config

```
option_settings:
  - namespace:  aws:elasticbeanstalk:container:tomcat:jvmoptions
    option_name:  Xmx
    value:  256m
  - option_name: MYPARAMETER
    value: parametervalue
```

簡易形式では、以下のようになります。

Example .ebextensions/options.config

```
option_settings:
  aws:elasticbeanstalk:container:tomcat:jvmoptions:
    Xmx: 256m
  aws:elasticbeanstalk:application:environment:
```

```
    MYPARAMETER: parametervalue
```

JSON の場合:

Example .ebextensions/options.config

```
{
  "option_settings": [
    {
      "namespace": "aws:elasticbeanstalk:container:tomcat:jvmoptions",
      "option_name": "Xmx",
      "value": "256m"
    },
    {
      "option_name": "MYPARAMETER",
      "value": "parametervalue"
    }
  ]
}
```

Linux サーバーでのソフトウェアのカスタマイズ

このセクションでは、Linux を実行する EC2 インスタンス上のソフトウェアをカスタマイズするための設定ファイルに含めることのできる情報の種類について説明します。Elastic Beanstalk 環境のカスタマイズと設定に関する一般的な情報については、「Elastic Beanstalk 環境の設定」を参照してください。Windows を実行する EC2 インスタンス上のソフトウェアをカスタマイズする方法については、Windows サーバーでのソフトウェアのカスタマイズを参照してください。

アプリケーションが依存するソフトウェアをカスタマイズして設定できます。インスタンスのプロビジョニング中に実行するコマンドを追加できます。たとえば、Linux ユーザーとグループの定義、環境インスタンスでのファイルのダウンロード、またはファイルの直接作成ができます。このようなファイルには、アプリケーションで必要な依存関係 (yum リポジトリからの追加パッケージなど) や、Elastic Beanstalk によるデフォルトの特定の設定を上書きするプロキシ構成ファイルの代わりとなる設定ファイルなどがあります。

> **メモ**
>
> - Amazon Linux 2 プラットフォームでは、.ebextensions 設定ファイルでファイルとコマンドを提供する代わりに、可能な限り Buildfile、Procfile、およびプラットフォームフックを使用して、インスタンスのプロビジョニング中に環境インスタンスでカスタムコードを設

定および実行することを強くお勧めします。これらのメカニズムの詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

- YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

設定ファイルは、アプリケーションが実行している Linux サーバーに影響する次のキーをサポートしています。

キー

- パッケージ
- グループ
- [ユーザー]
- [Sources] (出典)
- ファイル
- コマンド
- サービス
- コンテナコマンド
- 例: カスタム Amazon CloudWatch メトリクスの使用

キーは、上に表示した順に処理されます。

設定ファイルの作成およびテスト中は、環境のイベントを監視します。Elastic Beanstalk は、無効なキーなどの検証エラーが含まれている設定ファイルを無視し、同じファイルに含まれている他のキーを一切処理しません。これが発生すると、Elastic Beanstalk は警告イベントをイベントログに追加します。

パッケージ

packages キーを使用して、パッケージ済みのアプリケーションとコンポーネントをダウンロードしてインストールできます。

構文

```
packages:
  name of package manager:
```

```
    package name: version
    ...
  name of package manager:
    package name: version
    ...
  ...
```

各パッケージマネージャのキーで複数のパッケージを指定できます。

サポートされるパッケージ形式

現在 Elastic Beanstalk がサポートしているパッケージマネージャは、yum、rubygems、python、および rpm です。パッケージは rpm、yum、rubygems、python の順序で処理されます。rubygems と python の間に順序はありません。各パッケージマネージャ内では、パッケージのインストール順序は保証されません。オペレーティングシステムでサポートされているパッケージマネージャを使用します。

> **Note**
>
> Elastic Beanstalk は、Python の基盤となるパッケージマネージャとして pip と easy_install をサポートしています。ただし、設定ファイルの構文では、パッケージマネージャ名を `python` にする必要があります。設定ファイルを使用して Python のパッケージマネージャを指定すると、Elastic Beanstalk は Python 2.7 を使用します。アプリケーションが別のバージョンの Python に依存する場合は、インストールするパッケージを `requirements.txt` ファイルで指定できます。詳細については、「Elastic Beanstalk での要件ファイルを使用した依存関係の指定」を参照してください。

バージョンの指定

各パッケージマネージャ内では、各パッケージはパッケージ名およびバージョンのリストとして指定されます。バージョンは、文字列、バージョンのリスト、あるいは空の文字列またはリストのいずれでもかまいません。空の文字列またはリストは、最新バージョンを指定することを示します。rpm マネージャの場合、バージョンはディスク上のファイルへのパスまたは URL として指定します。相対パスはサポートされていません。

パッケージのバージョンを指定した場合は、それより新しいバージョンのパッケージがインスタンスに既にインストールされていたとしても、指定されたバージョンのインストールが試みられます。新しいバージョンが既にインストールされていた場合、デプロイは失敗します。パッケージマネージャには、複数のバージョンをサポートするものと、サポートしないものがあります。詳細については、

パッケージマネージャのドキュメントを調べてください。バージョンを指定せず、あるバージョンのパッケージが既にインストールされている場合は、Elastic Beanstalk は新しいバージョンをインストールせず、ユーザーが既存のバージョンを維持して使用することを望んでいるものと想定します。

例

次の例では、rpm のバージョン URL を指定し、yum から最新のバージョンを要求し、rubygems から chef のバージョン 0.10.2 を要求しています。

```
packages:
  yum:
    libmemcached: []
    ruby-devel: []
    gcc: []
  rpm:
    epel: http://download.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
  rubygems:
    chef: '0.10.2'
```

グループ

groups キーを使用すると、Linux/UNIX グループを作成して、グループ ID を割り当てることができます。グループを作成するには、新しいグループ名をオプションのグループ ID に関連付ける新しいキーと値のペアを追加します。groups キーでは、1 つまたは複数のグループ名を指定できます。次の表では使用できるキーの一覧を示します。

構文

```
groups:
  name of group: {}
  name of group:
    gid: "group id"
```

オプション

gid

グループ ID 番号です。

グループ ID を指定し、同じ名前のグループが既に存在する場合、グループの作成は失敗します。指定したグループ ID が別のグループに割り当てられている場合、オペレーティングシステムはグループの作成を拒否することがあります。

例

次の例では、groupOne という名前のグループをグループ ID なしで指定し、groupTwo という名前のグループをグループ ID 値 45 で指定しています。

```
groups:
  groupOne: {}
  groupTwo:
    gid: "45"
```

[ユーザー]

`users` キーを使用すると、Linux/UNIX ユーザーを EC2 インスタンス上に作成できます。

構文

```
users:
  name of user:
    groups:
      - name of group
    uid: "id of the user"
    homeDir: "user's home directory"
```

オプション

`uid`

ユーザー ID です。異なるユーザー ID で同じユーザー名が存在した場合、作成処理は失敗します。ユーザー ID が既存のユーザーに既に割り当てあれている場合、オペレーティングシステムは作成要求を拒否することがあります。

`groups`

グループ名のリストです。ユーザーはリスト内の各グループに追加されます。

`homeDir`

ユーザーのホームディレクトリです。

ユーザーは、`/sbin/nologin` のシェルで非対話形式のシステムユーザーとして作成されます。これは設計によるものであり、変更できません。

例

```
users:
  myuser:
    groups:
      - group1
      - group2
    uid: "50"
    homeDir: "/tmp"
```

[Sources] (出典)

`sources` キーを使用すると、公開 URL からアーカイブファイルをダウンロードし、EC2 インスタンス上のターゲットディレクトリに解凍できます。

構文

```
sources:
  target directory: location of archive file
```

サポートされる形式

サポートされる形式は、tar、tar+gzip、tar+bz2、zip です。Amazon Simple Storage Service (Amazon S3) などの外部の場所 (`https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject` など) は、その URL がパブリックアクセス可能であれば参照できます。

例

以下の例では、Amazon S3 バケットから公開 .zip ファイルをダウンロードし、`/etc/myapp` に解凍しています。

```
sources:
  /etc/myapp: https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject
```

> **Note**
>
> 複数の抽出で同じターゲットパスを再利用しないでください。別のソースを同じターゲットパスに抽出すると、コンテンツに追加する代わりに置き換えます。

ファイル

`files` キーを使用すると、EC2 インスタンス上にファイルを作成できます。内容は、設定ファイルにおいてインラインで指定することも、URL から取得することもできます。ファイルは辞書式順序でディスクに書き込まれます。

`files` キーを使用し、承認用のインスタンスプロファイルを指定することで、Amazon S3 からプライベートファイルをダウンロードできます。

指定したファイルパスが既にインスタンスに存在する場合、既存のファイルは名前に拡張子 `.bak` が付加されたまま保持されます。

構文

```
files:
  "target file location on disk":
     mode: "six-digit octal value"
     owner: name of owning user for file
     group: name of owning group for file
     source: URL
     authentication: authentication name:

  "target file location on disk":
     mode: "six-digit octal value"
     owner: name of owning user for file
     group: name of owning group for file
     content: |
      # this is my
      # file content
     encoding: encoding format
     authentication: authentication name:
```

オプション

`content`

ファイルに追加する文字列コンテンツ。`content` または `source` を指定します。両方を指定することはできません。

`source`

ダウンロードするファイルの URL。`content` または `source` を指定します。両方を指定することはできません。

`encoding`

content オプションで指定された文字列のエンコード形式。

有効な値: plain | base64

`group`

ファイルを所有している Linux グループ。

`owner`

ファイルを所有している Linux ユーザー。

`mode`

このファイルのモードを表す 6 桁の 8 進値です。Windows システムではサポートされていません。最初の 3 桁はシンボリックリンクに使用し、最後の 3 桁は権限の設定に使用します。シンボリックリンクを作成するには、120*xxx* を指定します (xxx はターゲットファイルのアクセス許可を定義します)。ファイルのアクセス許可を指定するには、「000644」のように後半 3 桁の数字を使用します。

`authentication`

使用する AWS CloudFormation 認証方法の名前。リソースキーで、Auto Scaling グループメタデータに認証方法を追加できます。例については、以下を参照してください。

例

```
files:
  "/home/ec2-user/myfile" :
    mode: "000755"
    owner: root
    group: root
    source: http://foo.bar/myfile

  "/home/ec2-user/myfile2" :
    mode: "000755"
    owner: root
    group: root
    content: |
      this is my
      file content
```

シンボリックリンクを使用する例。これは、既存のファイル /tmp/myfile2.txt を指すリンク /tmp/myfile1.txt を作成します。

```
files:
  "/tmp/myfile2.txt" :
    mode: "120400"
    content: "/tmp/myfile1.txt"
```

次の例では、リソースキーを使用して S3Auth という認証方法を追加して、Amazon S3 バケットから秘密ファイルをダウンロードするのに使用します。

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["amzn-s3-demo-bucket2"]
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"

files:
  "/tmp/data.json" :
    mode: "000755"
    owner: root
    group: root
    authentication: "S3Auth"
    source: https://elasticbeanstalk-us-west-2-123456789012.s3-us-west-2.amazonaws.com/
data.json
```

コマンド

commands キーを使用すると、EC2 インスタンスでコマンドを実行できます。このコマンドは、アプリケーションとウェブサーバーがセットアップされ、アプリケーションバージョンファイルが抽出される前に実行されます。

指定されたコマンドはルートユーザーとして実行され、名前のアルファベット順に処理されます。デフォルトでは、コマンドはルートディレクトリで実行します。別のディレクトリからコマンドを実行するには、cwd オプションを使用します。

コマンドの問題をトラブルシューティングするには、インスタンスログで出力を確認できます。

構文

```
commands:
  command name:
    command: command to run
    cwd: working directory
    env:
      variable name: variable value
    test: conditions for command
    ignoreErrors: true
```

オプション

command

実行するコマンドを指定する配列 (YAML シンタックスのブロックシーケンスコレクション) または文字列。重要な注意点:

- 文字列を使用する場合は、文字列全体を引用符で囲む必要はありません。引用符で囲む場合は、同じタイプの引用符がリテラルとして出現した場合にエスケープします。
- 配列を使用する場合、スペース文字をエスケープしたり、コマンドパラメータを引用符で囲んだりする必要はありません。配列の各要素が 1 つのコマンド引数です。配列を使用して複数のコマンドを指定しないでください。

以下の例はすべて同等です。

```
commands:
  command1:
    command: git commit -m "This is a comment."
  command2:
    command: "git commit -m \"This is a comment.\""
  command3:
    command: 'git commit -m "This is a comment."'
  command4:
    command:
      - git
      - commit
      - -m
      - This is a comment.
```

複数のコマンドを指定するには、次の例に示すように、リテラルブロックスカラーを使用します。

```
commands:
  command block:
    command: |
      git commit -m "This is a comment."
      git push
```

`env`

（オプション）コマンドの環境変数を設定します。このプロパティは、既存の環境に追加するのではなく、既存の環境を上書きします。

`cwd`

（オプション）作業ディレクトリです。指定されていない場合、コマンドはルート ディレクトリ (/) から実行します。

`test`

(オプション) Elastic Beanstalk が `command` キーに含まれるコマンド (シェルスクリプトなど) を処理するために、値 `true` (終了コード 0) を返す必要があるコマンドです。

`ignoreErrors`

（オプション）`command` キーに含まれるコマンドが失敗した場合（非ゼロ値を返した場合）、他のコマンドを実行する必要があるかどうかを指定するブール値です。失敗したコマンドがあっても他のコマンドの実行を続ける場合は、この値を `true` に設定します。コマンドが失敗したら実行を停止する場合は、`false` に設定します。デフォルト値は `false` です。

例

次の例では Python スクリプトを実行します。

```
commands:
  python_install:
    command: myscript.py
    cwd: /home/ec2-user
    env:
      myvarname: myvarvalue
    test: "[ -x /usr/bin/python ]"
```

サービス

`services` キーを使用すると、インスタンスが起動されるときに開始または停止する必要のあるサービスを定義できます。また、`services` キーではソース、パッケージ、ファイルへの依存関係も指定でき、インストールされているファイルのために再起動が必要になった場合に、Elastic Beanstalk がサービスの再起動を処理します。

構文

```
services:
  sysvinit:
    name of service:
      enabled: "true"
      ensureRunning: "true"
      files:
        - "file name"
      sources:
        - "directory"
      packages:
        name of package manager:
          "package name[: version]"
      commands:
        - "name of command"
```

オプション

`ensureRunning`

`true` に設定すると、Elastic Beanstalk が終了した後でサービスが実行されます。

`false` に設定すると、Elastic Beanstalk が終了した後でサービスは実行されません。

このキーを省略すると、サービスの状態は変更されません。

`enabled`

`true` に設定すると、起動時にサービスが自動的に開始されます。

`false` に設定すると、起動時にサービスが自動的に開始されません。

このキーを省略すると、このプロパティは変更されません。

`files`

ファイルのリストです。Elastic Beanstalk がファイルブロックによって直接変更した場合、サービスは再起動されます。

`sources`

ディレクトリのリストです。Elastic Beanstalk がこれらのディレクトリの 1 つにアーカイブを拡張した場合、サービスは再起動されます。

`packages`

パッケージ名のリストに対するパッケージマネージャのマップです。Elastic Beanstalk がこれらのパッケージの 1 つをインストールまたは更新した場合、サービスは再起動されます。

`commands`

コマンド名のリストです。Elastic Beanstalk が指定したコマンドを実行すると、サービスは再起動されます。

例

次に例を示します。

```
services:
  sysvinit:
    myservice:
      enabled: true
      ensureRunning: true
```

コンテナコマンド

`container_commands` キーを使用して、アプリケーションのソースコードに影響するコマンドを実行できます。コンテナコマンドは、アプリケーションおよびウェブサーバーが設定され、アプリケーションバージョンアーカイブが抽出された後、アプリケーションバージョンがデプロイされる前に実行されます。コンテナ以外のコマンドと他のカスタマイズオペレーションは、アプリケーションソースコードが抽出される前に実行されます。

指定されたコマンドはルートユーザーとして実行され、名前のアルファベット順に処理されます。コンテナコマンドは、ソースコードがアプリケーションサーバーにデプロイされる前に抽出されたステージングディレクトリから実行されます。コンテナコマンドを使用してステージングディレクトリにあるソースコードに対して行うすべての変更は、ソースが最終的な場所にデプロイされる際に含まれます。

> **Note**
>
> コンテナコマンドの出力は、`cfn-init-cmd.log` インスタンスログに記録されます。インスタンスログの取得と表示の詳細については、「Amazon EC2 インスタンスからのログの表示」を参照してください。

`leader_only` を使用して、1 つのインスタンスでコマンドを実行するか、テストコマンドが `test` と評価される場合のみコマンドを実行するよう `true` を設定できます。リーダー専用コンテナコマンドは、環境の作成およびデプロイ中のみ実行されます。他のコマンドやサーバーカスタマイズオペレーションは、インスタンスがプロビジョニングまたは更新されるたびに実行されます。リーダー専用コンテナコマンドは、AMI ID やインスタンスタイプの変更などの起動設定の変更によって実行されることはありません。

構文

```
container_commands:
  name of container_command:
    command: "command to run"
    leader_only: true
  name of container_command:
    command: "command to run"
```

オプション

`command`

実行する文字列または文字列の配列。

`env`

(オプション) コマンドを実行する前に環境変数を設定し、既存の値を上書きします。

`cwd`

(オプション) 作業ディレクトリです。デフォルトでは、これは解凍されたアプリケーションのステージングディレクトリです。

`leader_only`

(オプション) Elastic Beanstalk によって選択された単一のインスタンスでコマンドを実行するのみです。リーダー専用コンテナコマンドは、他のコンテナコマンドより前に実行されます。コマ

ンドはリーダー専用または test を持つことができますが、両方はできません (leader_only が優先されます)。

test

(オプション) このコンテナコマンドを実行するために、true を返す必要があるテストコマンドを実行します。コマンドはリーダー専用または test を持つことができますが、両方はできません (leader_only が優先されます)。

ignoreErrors

(オプション) このコンテナコマンドが 0 (成功) 以外の値を返す場合は、デプロイに失敗しません。有効にするには、true に設定します。

例

次に例を示します。

```
container_commands:
  collectstatic:
    command: "django-admin.py collectstatic --noinput"
  01syncdb:
    command: "django-admin.py syncdb --noinput"
    leader_only: true
  02migrate:
    command: "django-admin.py migrate"
    leader_only: true
  99customize:
    command: "scripts/customize.sh"
```

例: カスタム Amazon CloudWatch メトリクスの使用

このトピックでは、Amazon Linux 2 以降をベースとするプラットフォーム用の Amazon CloudWatch エージェントと Elastic Beanstalk メトリクスを統合する設定例を示します。設定例では、.ebextensions 設定ファイル内のファイルとコマンドを使用します。

Amazon CloudWatch は、さまざまなメトリクスをモニタリング、管理、発行し、メトリクスからのデータに基づいてアラームアクションを設定できるウェブサービスです。独自に使用するカスタムメトリクスを定義でき、Elastic Beanstalk はそのようなメトリクスを Amazon CloudWatch にプッシュします。Amazon CloudWatch に組み込まれたカスタムメトリクスは、Amazon CloudWatch コンソールで表示できます。

Amazon CloudWatch エージェント

Amazon CloudWatch エージェントは、オペレーティングシステム全体で Amazon EC2 インスタンスとオンプレミスサーバーの両方から CloudWatch メトリクスとログ収集を有効にします。エージェントは、システムレベルで収集されたメトリクスをサポートします。また、アプリケーションまたはサービスからカスタムログおよびメトリクスを収集できます。Amazon CloudWatch エージェントの詳細については、Amazon CloudWatch ユーザーガイドの CloudWatch エージェントを使用した メトリクスとログの収集を参照してください。

> **Note**
>
> Elastic Beanstalk の拡張ヘルスレポートでは、CloudWatch への幅広いインスタンスと環境メトリクスの発行がネイティブでサポートされています。詳細については、「環境の Amazon CloudWatch カスタムメトリクスの発行」を参照してください。

トピック

- .Ebextensions 設定ファイル
- アクセス許可
- CloudWatch コンソールでのメトリクスの表示

.Ebextensions 設定ファイル

この例では、.ebextensions 設定ファイルでファイルとコマンドを使用して、Amazon Linux 2 プラットフォームで Amazon CloudWatch エージェントを設定および実行しています。エージェントは Amazon Linux 2 で事前にパッケージ化されています。別のオペレーティングシステムを使用している場合は、エージェントをインストールするための追加手順が必要になる場合があります。詳細については、Amazon CloudWatch ユーザーガイドのCloudWatch エージェントのインストールを参照してください。

このサンプルを使用するには、プロジェクトディレクトリの最上位にある `.ebextensions` ディレクトリに `cloudwatch.config` というファイル名で保存した後、Elastic Beanstalk コンソール (ソースバンドルに .ebextensions ディレクトリを含める) または EB CLI を使用してアプリケーションをデプロイします。

設定ファイルの詳細については、「設定ファイル (`.ebextensions`) による高度な環境のカスタマイズ」を参照してください。

このファイルには 2 つのセクションがあります。

- `files` — このセクションでは、エージェント設定ファイルを追加します。エージェントが Amazon CloudWatch に送信するメトリクスとログを示します。この例では、mem_used_percent メトリクスのみを送信しています。Amazon CloudWatch エージェントでサポートされているシステムレベルのメトリクスの完全なリストについては、Amazon CloudWatch ユーザーガイドの CloudWatch エージェントにより収集されるメトリクスを参照してください。の。
- `container_commands` — このセクションには、エージェントを起動し、設定ファイルをパラメータとして渡すコマンドが含まれています。`container_commands` の詳細については、コンテナコマンド を参照してください。

.ebextensions/cloudwatch.config

```
files:
  "/opt/aws/amazon-cloudwatch-agent/bin/config.json":
    mode: "000600"
    owner: root
    group: root
    content: |
      {
        "agent": {
          "metrics_collection_interval": 60,
          "run_as_user": "root"
        },
        "metrics": {
          "namespace": "System/Linux",
          "append_dimensions": {
            "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
          },
          "metrics_collected": {
            "mem": {
              "measurement": [
                "mem_used_percent"
              ]
            }
          }
        }
      }
container_commands:
  start_cloudwatch_agent:
```

```
    command: /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a
append-config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json
```

アクセス許可

環境内のインスタンスには、Amazon CloudWatch エージェントを使用してカスタム Amazon CloudWatch メトリクスを発行するための適切な IAM アクセス許可が必要です。環境のインスタンスプロファイルにアクセス許可を追加することにより、その環境のインスタンスにアクセス許可を付与します。アプリケーションをデプロイする前または後のどちらでも、インスタンスプロファイルにアクセス権限を追加できます。

CloudWatch メトリックを発行する許可を付与するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [Roles] (ロール) を選択します。
3. 環境のインスタンスプロファイルのロールを選択します。デフォルトでは、Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合のロールは `aws-elasticbeanstalk-ec2-role` です。
4. [アクセス許可] タブを選択します。
5. [Permissions Policies] (アクセス許可ポリシー) の[Permissions] (アクセス許可) セクションで、[Attach policies] (ポリシーの添付) を選択します。
6. アクセス許可のアタッチで、 AWS 管理ポリシー CloudWatchAgentServerPolicy を選択します。次に、[Attach Policy] (ポリシーを添付) をクリックします。

ポリシーの管理の詳細については、IAM ユーザーガイドの「Working with Policies (ポリシーの使用)」を参照してください。

CloudWatch コンソールでのメトリクスの表示

CloudWatch の設定ファイルを環境にデプロイした後、Amazon CloudWatch コンソールを確認してメトリクスを表示します。カスタム指標は、CWAgent名前空間に配置されます。

詳細については、Amazon CloudWatch ユーザーガイドの「使用可能なメトリクスの表示」を参照してください。

Windows サーバーでのソフトウェアのカスタマイズ

アプリケーションが依存するソフトウェアをカスタマイズして設定できます。これらのファイルは、例えば、実行する必要のある追加パッケージやサービスなど、アプリケーションが必要とするいずれ

かの依存関係です。Elastic Beanstalk 環境のカスタマイズと設定に関する一般的な情報については、「Elastic Beanstalk 環境の設定」を参照してください。

> **Note**
>
> YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

設定ファイルは、アプリケーションが実行している Windows サーバーに影響する次のキーをサポートします。

キー

- パッケージ
- [Sources] (出典)
- ファイル
- コマンド
- サービス
- コンテナコマンド

キーは、上に表示した順に処理されます。

> **Note**
>
> 古い (バージョニング非対応) .NET プラットフォームのバージョンの場合は、正しい順序で設定ファイルが処理されません。詳細については、「Elastic Beanstalk Windows サーバープラットフォームのメジャーバージョン間での移行」を参照してください。

設定ファイルの作成およびテスト中は、環境のイベントを監視します。Elastic Beanstalk は、無効なキーなどの検証エラーが含まれている設定ファイルを無視し、同じファイルに含まれている他のキーを一切処理しません。これが発生すると、Elastic Beanstalk は警告イベントをイベントログに追加します。

パッケージ

packages キーを使用して、パッケージ済みのアプリケーションとコンポーネントをダウンロードしてインストールします。

Windows 環境では、Elastic Beanstalk は MSI パッケージのダウンロードとインストールをサポートしています。(Linux 環境は、追加のパッケージマネージャをサポートします。詳細については、Linux サーバーでのソフトウェアのカスタマイズページで パッケージ を参照してください)。

Amazon Simple Storage Service (Amazon S3) オブジェクトなどの外部の場所は、その URL がパブリックアクセス可能であれば参照できます。

複数の msi: パッケージを指定する場合、それらのインストール順序は保証されません。

構文

パッケージ名として選択した名前と、値としての MSI ファイルの場所への URL を指定します。msi: キーの下に複数のパッケージを指定できます。

```
packages:
  msi:
    package name: package url
    ...
```

例

次の例では、https://dev.mysql.com/ から [mysql] をダウンロードする URL を指定します。

```
packages:
  msi:
    mysql: https://dev.mysql.com/get/Downloads/Connector-Net/mysql-connector-
net-8.0.11.msi
```

次の例では、MSI ファイルの場所として Amazon S3 オブジェクトを指定しています。

```
packages:
  msi:
    mymsi: https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject.msi
```

[Sources] (出典)

`sources` キーを使用して、公開 URL からアーカイブファイルをダウンロードし、EC2 インスタンス上のターゲットディレクトリに解凍します。

構文

```
sources:
  target directory: location of archive file
```

サポートされる形式

Windows 環境では、Elastic Beanstalk は .zip 形式をサポートしています。(Linux 環境は、追加の形式をサポートします。詳細については、Linux サーバーでのソフトウェアのカスタマイズページで [Sources] (出典) を参照してください)。

Amazon Simple Storage Service (Amazon S3) オブジェクトなどの外部の場所は、その URL がパブリックアクセス可能であれば参照できます。

例

以下の例では、Amazon S3 から公開 .zip ファイルをダウンロードし、`c:/myproject/myapp` に解凍しています。

```
sources:
  "c:/myproject/myapp": https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject.zip
```

ファイル

`files` キーを使用して、EC2 インスタンス上にファイルを作成できます。内容は、設定ファイルでインラインとするか、URL から指定できます。ファイルは辞書式順序でディスクに書き込まれます。Amazon S3 からプライベートファイルをダウンロードするには、認可のインスタンスプロファイルを指定します。

構文

```
files:
  "target file location on disk":
    source: URL
    authentication: authentication name:
```

```
  "target file location on disk":
    content: |
      this is my content
    encoding: encoding format
```

オプション

`content`

(オプション) 文字列。

`source`

(オプション) ファイルの読み込み元の URL。このオプションと content キーを一緒に指定することはできません。

`encoding`

(オプション) エンコード形式。このオプションは、指定されたコンテンツキーの値にのみ使用されます。デフォルト値は `plain` です。

有効な値: `plain` | `base64`

`authentication`

(オプション) 使用する AWS CloudFormation 認証方法の名前。リソースキーで、Auto Scaling グループメタデータに認証方法を追加できます。

例

次の例は、ファイルコンテンツを提供する 2 つの方法である、URL からと設定ファイルのインラインからの方法を示しています。

```
files:
  "c:\\targetdirectory\\targetfile.txt":
    source: http://foo.bar/myfile

  "c:/targetdirectory/targetfile.txt":
    content: |
      # this is my file
      # with content
```

> **Note**
>
> ファイルパスにバックスラッシュ (\) を使用する場合、前の例に示すようにこの前に別のバックスラッシュ (エスケープ文字) を置く必要があります。

次の例では、リソースキーを使用して S3Auth という認証方法を追加して、Amazon S3 から秘密ファイルをダウンロードするのに使用します。

```
files:
  "c:\\targetdirectory\\targetfile.zip":
    source: https://elasticbeanstalk-us-east-2-123456789012.s3.amazonaws.com/prefix/
myfile.zip
    authentication: S3Auth

Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          s: ["amzn-s3-demo-bucket"]
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"
```

コマンド

commands キーを使用して、EC2 インスタンスでコマンドを実行します。コマンドは、名前のアルファベット順に処理され、アプリケーションとウェブサーバーが設定されてアプリケーションバージョンファイルが抽出される前に実行されます。

指定されたコマンドが管理者ユーザーとして実行されます。

コマンドの問題をトラブルシューティングするには、インスタンスログで出力を確認できます。

構文

```
commands:
  command name:
```

```
    command: command to run
```

オプション

`command`

実行するコマンドを指定する配列または文字列です。配列を使用する場合、スペース文字をエスケープしたり、コマンドパラメータを引用符で囲んだりする必要はありません。

`cwd`

（オプション）作業ディレクトリです。デフォルトでは、Elastic Beanstalk はプロジェクトのディレクトリの場所を探します。見つからない場合は、デフォルトとして `c:\Windows\System32` を使用します。

`env`

（オプション）コマンドの環境変数を設定します。このプロパティは、既存の環境に追加するのではなく、既存の環境を上書きします。

`ignoreErrors`

（オプション）`command` キーに含まれるコマンドが失敗した場合（非ゼロ値を返した場合）、他のコマンドを実行する必要があるかどうかを指定するブール値です。失敗したコマンドがあっても他のコマンドの実行を続ける場合は、この値を `true` に設定します。コマンドが失敗したら実行を停止する場合は、`false` に設定します。デフォルト値は `false` です。

`test`

(オプション) Elastic Beanstalk が `command` キーに含まれるコマンドを処理するために、値 `true` (終了コード 0) を返す必要があるコマンド。

`waitAfterCompletion`

(オプション) コマンドが完了してから次のコマンドを実行するまでに、待機する秒数。デフォルト値は **60** 秒です。を指定することもできます**forever**。

> **Important**
>
> デプロイ中のシステム再起動はサポートされていません。コマンドの結果としてシステムが再起動すると、インスタンスの初期化が失敗し、デプロイが失敗します。
> 回避策として、これを使用して`.ebextensions configuration`、デプロイの完了後に再起動をスケジュールできます。

例

次の例は、指定されたファイルに `set` コマンドの出力を保存します。後続のコマンドがある場合、Elastic Beanstalk は、このコマンドの完了直後にそのコマンドを実行します。このコマンドで再起動が必要な場合、Elastic Beanstalk はコマンドが完了すると、直後にインスタンスを再起動します。

```
commands:
  test:
    command: set > c:\\myapp\\set.txt
    waitAfterCompletion: 0
```

サービス

`services` キーを使用して、インスタンスが起動されるときに開始または停止する必要のあるサービスを定義できます。また、`services` キーではソース、パッケージ、ファイルへの依存関係も指定でき、インストールされているファイルのために再起動が必要になった場合に、Elastic Beanstalk がサービスの再起動を処理します。

構文

```
services:
  windows:
    name of service:
      files:
        - "file name"
      sources:
        - "directory"
      packages:
        name of package manager:
            "package name[: version]"
      commands:
        - "name of command"
```

オプション

`ensureRunning`

(オプション) `true` に設定すると、Elastic Beanstalk が終了した後でサービスが実行されます。

`false` に設定すると、Elastic Beanstalk が終了した後でサービスは実行されません。

このキーを省略すると、サービスの状態は変更されません。

`enabled`

(オプション) `true` に設定すると、起動時にサービスが自動的に開始されます。

`false` に設定すると、起動時にサービスが自動的に開始されません。

このキーを省略すると、このプロパティは変更されません。

`files`

ファイルのリストです。Elastic Beanstalk がファイルブロックによって直接変更した場合、サービスは再起動されます。

`sources`

ディレクトリのリストです。Elastic Beanstalk がこれらのディレクトリの 1 つにアーカイブを拡張した場合、サービスは再起動されます。

`packages`

パッケージ名のリストに対するパッケージマネージャのマップです。Elastic Beanstalk がこれらのパッケージの 1 つをインストールまたは更新した場合、サービスは再起動されます。

`commands`

コマンド名のリストです。Elastic Beanstalk が指定したコマンドを実行すると、サービスは再起動されます。

例

```
services:
  windows:
    myservice:
      enabled: true
      ensureRunning: true
```

コンテナコマンド

`container_commands` キーを使用して、アプリケーションのソースコードに影響するコマンドを実行します。コンテナコマンドは、アプリケーションおよびウェブサーバーが設定され、アプリケーションバージョンアーカイブが抽出された後、アプリケーションバージョンがデプロイされる前に実行されます。コンテナ以外のコマンドと他のカスタマイズオペレーションは、アプリケーションソースコードが抽出される前に実行されます。

コンテナコマンドは、ソースコードがアプリケーションサーバーにデプロイされる前に抽出されたステージングディレクトリから実行されます。コンテナコマンドを使用してステージングディレクトリにあるソースコードに対して行うすべての変更は、ソースが最終的な場所にデプロイされる際に含まれます。

コンテナコマンドの問題をトラブルシューティングするには、インスタンスログで出力を確認できます。

`leader_only` オプションを使用して、1 つのインスタンスでコマンドを実行するか、テストコマンドが `test` と評価される場合のみコマンドを実行するよう `true` を設定します。リーダー専用コンテナコマンドは、環境の作成およびデプロイ中のみ実行されます。他のコマンドやサーバーカスタマイズオペレーションは、インスタンスがプロビジョニングまたは更新されるたびに実行されます。リーダー専用コンテナコマンドは、AMI ID やインスタンスタイプの変更などの起動設定の変更によって実行されることはありません。

構文

```
container_commands:
  name of container_command:
    command: command to run
```

オプション

`command`

実行する文字列または文字列の配列。

`env`

(オプション) コマンドを実行する前に環境変数を設定し、既存の値を上書きします。

`cwd`

(オプション) 作業ディレクトリです。デフォルトでは、これは解凍されたアプリケーションのステージングディレクトリです。

`leader_only`

(オプション) Elastic Beanstalk によって選択された単一のインスタンスでコマンドを実行するのみです。リーダー専用コンテナコマンドは、他のコンテナコマンドより前に実行されます。コマンドはリーダー専用または `test` を持つことができますが、両方はできません (`leader_only` が優先されます)。

`test`

(オプション) このコンテナコマンドを実行するために、`true` を返す必要があるテストコマンドを実行します。コマンドはリーダー専用または `test` を持つことができますが、両方はできません (`leader_only` が優先されます)。

`ignoreErrors`

(オプション) このコンテナコマンドが 0 (成功) 以外の値を返す場合は、デプロイに失敗しません。有効にするには、`true` に設定します。

`waitAfterCompletion`

(オプション) コマンドが完了してから次のコマンドを実行するまでに、待機する秒数。デフォルト値は **60** 秒です。を指定することもできます**forever**。

> **Important**
>
> デプロイ中のシステム再起動はサポートされていません。コマンドの結果としてシステムが再起動すると、インスタンスの初期化が失敗し、デプロイが失敗します。
> 回避策として、これを使用して`.ebextensions configuration`、デプロイの完了後に再起動をスケジュールできます。

例

次の例は、指定されたファイルに `set` コマンドの出力を保存します。Elastic Beanstalk は 1 つのインスタンスでコマンドを実行し、コマンドが完了すると、直後にインスタンスを再起動します。

```
container_commands:
  foo:
    command: set > c:\\myapp\\set.txt
    leader_only: true
    waitAfterCompletion: 0
```

Elastic Beanstalk 環境リソースの追加とカスタマイズ

また、Elastic Beanstalk 環境の一部である環境リソースを、カスタマイズする必要性が生じることもあります。例えば、Amazon SQS キューおよびキューの深さに対するアラームを追加したり、Amazon ElastiCache クラスターを追加したりする場合です。ソースバンドルと共に設定ファイ

ルを含めることにより、アプリケーションバージョンのデプロイと同時に環境を簡単にカスタマイズできます。

設定ファイルの Resourcesキーを使用して、 環境内の AWS リソースを作成およびカスタマイズできます。設定ファイルで定義されたリソースは、環境の起動に使用される AWS CloudFormation テンプレートに追加されます。すべての AWS CloudFormation リソースタイプがサポートされています。

> **Note**
>
> Elastic Beanstalk によって管理されていないリソースを追加する場合は、必ず適切なアクセス許可を持つユーザーポリシーを AWS Identity and Access Management (IAM) ユーザーに追加してください。Elastic Beanstalk が提供する管理ユーザーポリシーは、Elastic Beanstalk で管理されるリソースに対するアクセス許可のみを対象としています。

たとえば、次の設定ファイルは Auto Scaling ライフサイクルフックを Elastic Beanstalk によって作成されたデフォルトの Auto Scaling グループに追加します。

~/my-app/.ebextensions/as-hook.config

```
Resources:
  hookrole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument: {
               "Version" : "2012-10-17",
               "Statement": [ {
                  "Effect": "Allow",
                  "Principal": {
                     "Service": [ "autoscaling.amazonaws.com" ]
                  },
                  "Action": [ "sts:AssumeRole" ]
               } ]
            }
      Policies: [ {
               "PolicyName": "SNS",
               "PolicyDocument": {
                      "Version": "2012-10-17",
                      "Statement": [{
                          "Effect": "Allow",
```

```
                            "Resource": "*",
                            "Action": [
                                "sqs:SendMessage",
                                "sqs:GetQueueUrl",
                                "sns:Publish"
                            ]
                          }
                        ]
                    }
                } ]
  hooktopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint: "my-email@example.com"
          Protocol: email
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      AutoScalingGroupName: { "Ref" : "AWSEBAutoScalingGroup" }
      LifecycleTransition: autoscaling:EC2_INSTANCE_TERMINATING
      NotificationTargetARN: { "Ref" : "hooktopic" }
      RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn"] }
```

この例では、3 つのリソース (`hookrole`、`hooktopic` および `lifecyclehook`) を定義しています。最初の 2 つのリソースはメッセージを Amazon SNS に発行する許可を Amazon EC2 Auto Scaling に付与する IAM ロールと、Auto Scaling グループから E メールアドレスにメッセージをリレーする SNS トピックです。Elastic Beanstalk は、指定したプロパティとタイプを持つこれらのリソースを作成します。

最後のリソース `lifecyclehook` はライフサイクルフック自体です。

```
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      AutoScalingGroupName: { "Ref" : "AWSEBAutoScalingGroup" }
      LifecycleTransition: autoscaling:EC2_INSTANCE_TERMINATING
      NotificationTargetARN: { "Ref" : "hooktopic" }
      RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn"] }
```

ライフサイクルフックの定義では、2 つの関数を使用して、フックのプロパティに値を入力します。`{ "Ref" : "AWSEBAutoScalingGroup" }` は、環境の Elastic Beanstalk によって作成され

た Auto Scaling グループの名前を取得します。`AWSEBAutoScalingGroup` は Elastic Beanstalk によって提供される標準リソース名の 1 つです。

`AWS::IAM::Role` では、`Ref` が返すのは ARN ではなくロールの名前のみです。`RoleARN` パラメータの ARN を取得するには、代わりに別の組み込み関数 `Fn::GetAtt` を使用することで、リソースから属性を取得できます。`RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn"] }` は `hookrole` リソースから `Arn` 属性を取得します。

`{ "Ref" : "hooktopic" }` は、設定ファイルで前に作成した Amazon SNS トピックの ARN を取得します。によって返される値はリソースタイプごとにRef異なり、AWS::SNS::Topic リソースタイプの AWS CloudFormation ユーザーガイドトピックにあります。 AWS::SNS::Topic

Elastic Beanstalk が環境向けに作成するリソースを変更する

Elastic Beanstalk が環境向けに作成するリソースには、名前があります。これらの名前を使用して、関数でリソースに関する情報を取得するか、リソース上のプロパティを変更して動作をカスタマイズすることができます。このトピックでは、さまざまなタイプの環境で Elastic Beanstalk が使用する AWS リソースについて説明します。

> **Note**
>
> 前のトピックカスタムリソースでは、環境リソースをカスタマイズするためのいくつかの使用例と例を示します。また、設定ファイルのサンプルについては、後のトピック カスタムリソースの例 で参照できます。

ウェブサーバー環境には次のリソースがあります。

「ウェブサーバー環境」

- `AWSEBAutoScalingGroup` (AWS::AutoScaling::AutoScalingGroup) - 環境にアタッチされる Auto Scaling グループ。
- 次の 2 つのリソースのいずれか。
  - `AWSEBAutoScalingLaunchConfiguration` (AWS::AutoScaling::LaunchConfiguration) - 環境の Auto Scaling グループにアタッチされる起動設定。
  - `AWSEBEC2LaunchTemplate` (AWS::EC2::LaunchTemplate) - 環境の Auto Scaling グループが使用する Amazon EC2 起動テンプレート。

> **Note**
>
> 環境で、Amazon EC2 起動テンプレートを必要とする機能が使用されており、ユーザーポリシーに必要なアクセス許可がない場合、環境の作成または更新が失敗することがあります。管理ユーザーポリシーの AdministratorAccess-AWSElasticBeanstalk を使用するか、カスタムポリシーに必要な許可を追加するようにしてください。

- `AWSEBEnvironmentName` (AWS::ElasticBeanstalk::Environment) - 環境。
- `AWSEBSecurityGroup` (AWS::EC2::SecurityGroup) - Auto Scaling グループにアタッチされるセキュリティグループ。
- `AWSEBRDSDatabase` (AWS::RDS::DBInstance) - 環境にアタッチされる Amazon RDS DB インスタンス (該当する場合)。

負荷分散された環境では、ロードバランサーに関する追加のリソースにアクセスできます。標準ロードバランサーには、ロードバランサーのリソースと、それにアタッチされているセキュリティグループのリソースがあります。アプリケーションおよびnetwork load balancerには、ロードバランサーのデフォルトリスナー、リスナールール、およびターゲットグループの追加リソースがあります。

負荷分散された環境

- `AWSEBLoadBalancer` (AWS::ElasticLoadBalancing::LoadBalancer) - 環境の Classic Load Balancer。
- `AWSEBV2LoadBalancer` (AWS::ElasticLoadBalancingV2::LoadBalancer) - 環境の Application Load Balancer または Network Load Balancer。
- `AWSEBLoadBalancerSecurityGroup` (AWS::EC2::SecurityGroup) - カスタム Amazon Virtual Private Cloud (Amazon VPC) のみで、Elastic Beanstalk がロードバランサー用に作成するセキュリティグループの名前。デフォルトの VPC または EC2 classic では、Elastic Load Balancing はデフォルトのセキュリティグループをロードバランサーに割り当てます。
- `AWSEBV2LoadBalancerListener` (AWS::ElasticLoadBalancingV2::Listener) - ロードバランサーが、接続リクエストを確認し、1 つ以上のターゲットグループに転送することを許可するリスナー。
- `AWSEBV2LoadBalancerListenerRule` (AWS::ElasticLoadBalancingV2::ListenerRule) - Elastic Load Balancing リスナーがアクション実行するリクエストと、そのリクエストで実行されるアクションを定義します。

- `AWSEBV2LoadBalancerTargetGroup` (AWS::ElasticLoadBalancingV2::TargetGroup) - Amazon EC2 インスタンスなど、1つ 以上の登録済みターゲットにリクエストをルーティングする Elastic Load Balancing ターゲットグループ。

ワーカー環境には、受信リクエストをバッファリングする SQS キューのリソースと、インスタンスでリーダーの選択に使用する Amazon DynamoDB テーブルが含まれます。

ワーカー環境

- `AWSEBWorkerQueue` (AWS::SQS::Queue) - デーモンが処理する必要があるリクエストをプルする対象の Amazon SQS キュー。
- `AWSEBWorkerDeadLetterQueue` (AWS::SQS::Queue) - デーモンによって配信できなかったメッセージや、正常に処理できなかったその他のメッセージを保存する Amazon SQS キュー。
- `AWSEBWorkerCronLeaderRegistry` (AWS::DynamoDB::Table) - 定期的なタスクで使用される内部レジストリである Amazon DynamoDB テーブル。

その他の AWS CloudFormation テンプレートキー

`Resources`、、 AWS CloudFormation などの設定ファイルキーは `files`から既に導入されています`packages`。Elastic Beanstalk は、環境をサポートする AWS CloudFormation テンプレートに設定ファイルの内容を追加するため、他の AWS CloudFormation セクションを使用して設定ファイルで高度なタスクを実行できます。

キー

- パラメータ
- 出力
- Mappings

パラメータ

パラメータは、Elastic Beanstalk 独自の代替的なカスタムオプションであり、これを使用して、設定ファイルの他の場所で使用する値を定義することができます。カスタムオプションと同様、パラメータを使用して、1 つの場所でユーザーが設定可能な値を収集することができます。カスタムオプションとは異なり、Elastic Beanstalk の API を使用してパラメータ値を設定することはできず、テンプレートで定義できるパラメータの数は によって制限されます AWS CloudFormation。

パラメータを使用する理由の 1 つは、設定ファイルを AWS CloudFormation テンプレートとして 2 倍にすることです。カスタムオプションの代わりにパラメータを使用する場合は、設定ファイルを使用して、独自のスタック AWS CloudFormation と同じリソースを に作成できます。たとえば、テスト用環境に Amazon EFS ファイルシステムを追加する設定ファイルがある場合は、同一ファイルを使用して、本稼働用の環境ライフサイクルに固定されていない独自のファイルシステムを作成します。

以下の例では、設定ファイルの上部にあるユーザーが設定可能な値を収集するパラメータを示します。

Example loadbalancer-accesslogs-existingbucket.config - パラメータ

```
Parameters:
  bucket:
    Type: String
    Description: "Name of the Amazon S3 bucket in which to store load balancer logs"
    Default: "amzn-s3-demo-bucket"
  bucketprefix:
    Type: String
    Description: "Optional prefix. Can't start or end with a /, or contain the word
 AWSLogs"
    Default: ""
```

出力

`Outputs` ブロックを使用して、作成されたリソースに関する情報を AWS CloudFormationにエクスポートすることができます。その後、 `Fn::ImportValue`関数を使用して、Elastic Beanstalk 外の AWS CloudFormation テンプレートに値をプルできます。

次の例では、Amazon SNS トピックを作成し、その ARN を という名前 AWS CloudFormation で にエクスポートします`NotificationTopicArn`。

Example sns-topic.config

```
Resources:
  NotificationTopic:
    Type: AWS::SNS::Topic

Outputs:
  NotificationTopicArn:
    Description: Notification topic ARN
    Value: { "Ref" : "NotificationTopic" }
```

```
    Export:
      Name: NotificationTopicArn
```

別の環境の設定ファイル、または Elastic Beanstalk 外の AWS CloudFormation テンプレートでは、`Fn::ImportValue`関数を使用してエクスポートされた ARN を取得できます。この例では、エクスポートされた値を `TOPIC_ARN` という名前の環境プロパティに割り当てます。

Example env.config

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    TOPIC_ARN: '`{ "Fn::ImportValue" : "NotificationTopicArn" }`'
```

Mappings

マッピングを使用して、名前空間ごとに分けられたキーと値のペアを保存することができます。マッピングでは、Config を通して値を整理したり、その他の値に応じてパラメータ値を変更することができます。たとえば、次の設定値では、現在のリージョンに基づいて、アカウント ID のパラメータが設定されます。

Example [loadbalancer-accesslogs-newbucket.config](#) - マッピング

```
Mappings:
  Region2ELBAccountId:
    us-east-1:
      AccountId: "111122223333"
    us-west-2:
      AccountId: "444455556666"
    us-west-1:
      AccountId: "123456789012"
    eu-west-1:
      AccountId: "777788889999"
...
            Principal:
              AWS:
                ? "Fn::FindInMap"
                :
                  - Region2ELBAccountId
                  -
                    Ref: "AWS::Region"
                  - AccountId
```

関数

設定ファイル内の関数を使用して、別のリソースまたは Elastic Beanstalk 設定オプションの設定からの情報で、リソースプロパティの値を入力できます。Elastic Beanstalk は、 AWS CloudFormation 関数 (`Ref`、`Fn::GetAtt`、`Fn::Join`) と 1 つの Elastic Beanstalk 固有の関数 をサポートしていますFn::GetOptionSetting。

関数

- 参照番号
- Fn::GetAtt
- Fn::Join
- Fn::GetOptionSetting

参照番号

を使用してRef、 AWS リソースのデフォルトの文字列表現を取得します。Ref によって返される値は、リソースタイプによって異なりますが、他の要因によって異なる場合もあります。たとえば、セキュリティグループ (AWS::EC2::SecurityGroup) は、セキュリティグループがデフォルト Amazon Virtual Private Cloud (Amazon VPC)、EC2 Classic、カスタム VPC 内のいずれにあるかに応じて、セキュリティグループの名前または ID を返します。

```
{ "Ref" : "resource name" }
```

> **Note**
>
> Ref の戻り値を含む各リソースタイプの詳細については、AWS ユーザーガイドの、AWS CloudFormation リソースプロパティタイプのリファレンスを参照してください。

サンプルの Auto Scaling のライフサイクルフックから:

```
Resources:
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      AutoScalingGroupName: { "Ref" : "AWSEBAutoScalingGroup" }
```

`Ref` を使用して、同じファイルまたは別の設定ファイルの他の場所で定義された AWS CloudFormation パラメータの値を取得することもできます。

Fn::GetAtt

を使用して`Fn::GetAtt`、 AWS リソースの属性の値を取得します。

```
{ "Fn::GetAtt" : [ "resource name", "attribute name"] }
```

サンプルの Auto Scaling のライフサイクルフックから:

```
Resources:
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn"] }
```

詳細については、「Fn::GetAtt」を参照してください。

Fn::Join

`Fn::Join` を使用して、区切り文字の文字列を結合します。文字列はハードコード化したり、`Fn::GetAtt` または `Ref` の出力を使用できます。

```
{ "Fn::Join" : [ "delimiter", [ "string1", "string2" ] ] }
```

詳細については、「Fn::Join」を参照してください。

Fn::GetOptionSetting

`Fn::GetOptionSetting` を使用して、環境に適用する設定オプションの設定の値を取得します。

```
"Fn::GetOptionSetting":
  Namespace: "namespace"
  OptionName: "option name"
  DefaultValue: "default value"
```

プライベートキーを保存からの例

```
Resources:
  AWSEBAutoScalingGroup:
```

```
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["elasticbeanstalk-us-west-2-123456789012"]
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"
```

カスタムリソースの例

次に示すのは、Elastic Beanstalk 環境のカスタマイズに使用できる設定ファイルの例のリストです。

- DynamoDB、CloudWatch、および SNS
- Elastic Load Balancing および CloudWatch
- ElastiCache
- RDS および CloudWatch
- SQS、SNS、および CloudWatch

このページのサブトピックでは、Elastic Beanstalk 環境でカスタムリソースを追加および設定するための拡張例をいくつか示します。

例

- 例: ElastiCache
- 例: SQS、CloudWatch、SNS
- 例: DynamoDB、CloudWatch、SNS

例: ElastiCache

以下のサンプルでは、Amazon ElastiCache クラスターを EC2-Classic および EC2-VPC (デフォルトおよびカスタム Amazon Virtual Private Cloud (Amazon VPC) の両方) プラットフォームに追加します。これらのプラットフォームの詳細と、EC2 がリージョンと AWS アカウントでサポートするプラットフォームを特定する方法については、「」を参照してくださいhttps://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-supported-platforms.html。このトピックの、使用しているプラットフォームに該当するセクションを参照してください。

- EC2-classic プラットフォーム
- EC2-VPC (デフォルト)
- EC2-VPC (カスタム)

EC2-classic プラットフォーム

このサンプルは、EC2-Classic プラットフォーム内にインスタンスが起動される環境に Amazon ElastiCache クラスターを追加します。この例で示されているすべてのプロパティは、各リソースタイプに対して設定する必要のある最低限必要なプロパティです。サンプルは、「ElastiCache example」でダウンロードできます。

> **Note**
>
> この例では、課金される可能性のある AWS リソースを作成します。 AWS 料金の詳細については、「」を参照してくださいhttps://aws.amazon.com/pricing/。一部の サービスは AWS 無料利用枠の一部です。新規のお客様は、無料でこれらのサービスをテストできる場合があります。詳細については「https://aws.amazon.com/free/」を参照してください。

この例を使用するには、以下を実行します。

1. ソースバンドルの最上位ディレクトリに `.ebextensions` ディレクトリを作成します。
2. 拡張子が `.config` の設定ファイルを 2 つ作成し、`.ebextensions` ディレクトリに配置します。一方の設定ファイルではリソースが定義され、もう一方の設定ファイルではオプションが定義されます。
3. アプリケーションを Elastic Beanstalk にデプロイします。

   YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

リソースを定義する設定ファイル (例: `elasticache.config`) を作成します。最初に、ElastiCache クラスターリソースの名前 (`MyElastiCache`) を指定し、タイプを宣言し、クラスターのプロパティを設定することによって、ElastiCache クラスターを作成します。この例は、この設定ファイルで作成され、定義されている ElastiCache セキュリティグループリソースの名前を参照します。次に、ElastiCache セキュリティグループを作成します。このリソースの名前を定義し、

タイプを宣言し、セキュリティグループの説明を追加します。最後に、ElastiCache セキュリティグループの入口ルールを設定し、ElastiCache セキュリティグループ（`MyCacheSecurityGroup`）および Elastic Beanstalk セキュリティグループ（`AWSEBSecurityGroup`）の内部のインスタンスからのアクセスのみを許可するようにします。パラメータ名 `AWSEBSecurityGroup` は、Elastic Beanstalk によって提供される固定のリソース名です。Elastic Beanstalk アプリケーションから ElastiCache クラスター内のインスタンスに接続するには、ElastiCache セキュリティグループの入口ルールに `AWSEBSecurityGroup` を追加する必要があります。

```
#This sample requires you to create a separate configuration file that defines the
 custom option settings for CacheCluster properties.

Resources:
  MyElastiCache:
    Type: AWS::ElastiCache::CacheCluster
    Properties:
      CacheNodeType:
         Fn::GetOptionSetting:
             OptionName : CacheNodeType
             DefaultValue: cache.m1.small
      NumCacheNodes:
           Fn::GetOptionSetting:
             OptionName : NumCacheNodes
             DefaultValue: 1
      Engine:
           Fn::GetOptionSetting:
             OptionName : Engine
             DefaultValue: memcached
      CacheSecurityGroupNames:
        - Ref: MyCacheSecurityGroup
  MyCacheSecurityGroup:
    Type: AWS::ElastiCache::SecurityGroup
    Properties:
      Description: "Lock cache down to webserver access only"
  MyCacheSecurityGroupIngress:
    Type: AWS::ElastiCache::SecurityGroupIngress
    Properties:
      CacheSecurityGroupName:
        Ref: MyCacheSecurityGroup
      EC2SecurityGroupName:
        Ref: AWSEBSecurityGroup
```

この設定ファイル例で使用されているリソースの詳細については、以下を参照してください。

- AWS::ElastiCache::CacheCluster
- AWS::ElastiCache::SecurityGroup
- AWS::ElastiCache::SecurityGroupIngress

options.config という名前の別の設定ファイルを作成し、カスタムオプションの設定を定義します。

```
option_settings:
  "aws:elasticbeanstalk:customoption":
     CacheNodeType : cache.m1.small
     NumCacheNodes : 1
     Engine : memcached
```

これらの行は、Elastic Beanstalk に対し、CacheNodeType、NumCacheNodes、Engine プロパティの値を、option_settings セクションと、使用する実際の値を含む名前と値のペアを含む aws:elasticbeanstalk:customoption セクションが含まれている設定ファイル (この例では options.config) の CacheNodeType、NumCacheNodes、Engine の値から取得するように指示します。上の例では、これは値として cache.m1.small、1、memcached を使用することを意味します。Fn::GetOptionSetting の詳細については、「関数」を参照してください。

EC2-VPC (デフォルト)

このサンプルは、EC2-VPC プラットフォーム内にインスタンスが起動される環境に Amazon ElastiCache クラスターを追加します。具体的には、このセクションの情報は、EC2 がデフォルト VPC 内にインスタンスを起動するシナリオに適用されます。この例のプロパティはすべて、リソースタイプごとに設定する必要のある最低限必要なプロパティです。デフォルト VPC の詳細については、「Your Default VPC and Subnets」を参照してください。

> **Note**
>
> この例では、課金される可能性のある AWS リソースを作成します。 AWS 料金の詳細については、「」を参照してくださいhttps://aws.amazon.com/pricing/。一部の サービスは AWS 無料利用枠の一部です。新規のお客様は、無料でこれらのサービスをテストできる場合があります。詳細については「https://aws.amazon.com/free/」を参照してください。

この例を使用するには、以下を実行します。

1. ソースバンドルの最上位ディレクトリに .ebextensions ディレクトリを作成します。

2. 拡張子が `.config` の設定ファイルを 2 つ作成し、`.ebextensions` ディレクトリに配置します。一方の設定ファイルではリソースが定義され、もう一方の設定ファイルではオプションが定義されます。
3. アプリケーションを Elastic Beanstalk にデプロイします。

   YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

ここで、リソースの設定ファイルに `elasticache.config` という名前を付けます。ElastiCache クラスターを作成するため、この例は ElastiCache クラスターリソースの名前 (`MyElastiCache`) を指定し、そのタイプを宣言してから、クラスターのプロパティを設定します。この例は、この設定ファイルで作成し、定義するセキュリティグループリソースの ID を参照します。

次に、EC2 セキュリティグループを作成します。このリソースの名前の定義、そのタイプの宣言、および説明の入力を行ってから、Elastic Beanstalk セキュリティグループ (`AWSEBSecurityGroup`) 内のインスタンスからのアクセスのみを許可するようにセキュリティグループの入口ルールを設定します (パラメータ名 `AWSEBSecurityGroup` は、Elastic Beanstalk から提供される固定のリソース名です。Elastic Beanstalk アプリケーションから ElastiCache クラスター内のインスタンスに接続するには、ElastiCache セキュリティグループの入口ルールに `AWSEBSecurityGroup`を追加する必要があります)。

EC2 セキュリティグループの入口ルールは、キャッシュノードが接続を受け付けることができる IP プロトコルとポート番号も定義します。Redis の場合、デフォルトのポート番号は 6379 です。

```
#This sample requires you to create a separate configuration file that defines the
 custom option settings for CacheCluster properties.

Resources:
  MyCacheSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: "Lock cache down to webserver access only"
      SecurityGroupIngress :
        - IpProtocol : "tcp"
          FromPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          ToPort :
```

```
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          SourceSecurityGroupName:
            Ref: "AWSEBSecurityGroup"
  MyElastiCache:
    Type: "AWS::ElastiCache::CacheCluster"
    Properties:
      CacheNodeType:
        Fn::GetOptionSetting:
          OptionName : "CacheNodeType"
          DefaultValue : "cache.t2.micro"
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName : "NumCacheNodes"
          DefaultValue : "1"
      Engine:
        Fn::GetOptionSetting:
          OptionName : "Engine"
          DefaultValue : "redis"
      VpcSecurityGroupIds:
        -
          Fn::GetAtt:
            - MyCacheSecurityGroup
            - GroupId

Outputs:
  ElastiCache:
    Description : "ID of ElastiCache Cache Cluster with Redis Engine"
    Value :
      Ref : "MyElastiCache"
```

この設定ファイル例で使用されているリソースの詳細については、以下を参照してください。

- AWS::ElastiCache::CacheCluster
- AWS::EC2::SecurityGroup

次に、オプションの設定ファイルに `options.config` という名前を付け、カスタムオプションの設定を定義します。

```
option_settings:
  "aws:elasticbeanstalk:customoption":
```

```
    CacheNodeType : cache.t2.micro
    NumCacheNodes : 1
    Engine : redis
    CachePort : 6379
```

このコードは、設定ファイル (この例では options.config) の CacheNodeType、NumCacheNodes、Engine、および CachePort の値から CacheNodeType、NumCacheNodes、Engine、および CachePort プロパティの値を取得するように Elastic Beanstalk に指示します。そのファイルには aws:elasticbeanstalk:customoption セクションが含まれており (option_settings に)、そこには実際に使用される値を示す名前と値のペアが含まれています。前述の例では、cache.t2.micro、1、redis、および 6379 が値として使用されます。Fn::GetOptionSetting の詳細については、「関数」を参照してください。

EC2-VPC (カスタム)

EC2-VPC プラットフォームにカスタム VPC を作成し、EC2 がインスタンスを起動する VPC として指定する場合、ユーザーの環境に Amazon ElastiCache クラスターを追加するプロセスはデフォルト VPC の場合とは異なります。主な違いは、ElastiCache クラスター用のサブネットグループを作成する必要があることです。この例のプロパティはすべて、リソースタイプごとに設定する必要のある最低限必要なプロパティです。

> **Note**
>
> この例では、課金される可能性のある AWS リソースを作成します。 AWS 料金の詳細については、「」を参照してくださいhttps://aws.amazon.com/pricing/。一部の サービスは AWS 無料利用枠の一部です。新規のお客様は、無料でこれらのサービスをテストできる場合があります。詳細については「https://aws.amazon.com/free/」を参照してください。

この例を使用するには、以下を実行します。

1. ソースバンドルの最上位ディレクトリに .ebextensions ディレクトリを作成します。
2. 拡張子が .config の設定ファイルを 2 つ作成し、.ebextensions ディレクトリに配置します。一方の設定ファイルではリソースが定義され、もう一方の設定ファイルではオプションが定義されます。
3. アプリケーションを Elastic Beanstalk にデプロイします。

YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

ここで、リソースの設定ファイルに `elasticache.config` という名前を付けます。ElastiCache クラスターを作成するため、この例は ElastiCache クラスターリソースの名前 (`MyElastiCache`) を指定し、そのタイプを宣言してから、クラスターのプロパティを設定します。この例のプロパティは、ElastiCache クラスター用のサブネットグループの名前だけではなく、この設定ファイルで作成し、定義したセキュリティグループリソースの ID も参照します。

次に、EC2 セキュリティグループを作成します。このリソースの名前の定義、そのタイプの宣言、説明の入力、および VPC ID の割り当てを行ってから、Elastic Beanstalk セキュリティグループ (`AWSEBSecurityGroup`) 内のインスタンスからのアクセスのみを許可するようにセキュリティグループの入口ルールを設定します (パラメータ名 `AWSEBSecurityGroup` は、Elastic Beanstalk から提供される固定のリソース名です。Elastic Beanstalk アプリケーションから ElastiCache クラスター内のインスタンスに接続するには、ElastiCache セキュリティグループの入口ルールに `AWSEBSecurityGroup`を追加する必要があります)。

EC2 セキュリティグループの入口ルールは、キャッシュノードが接続を受け付けることができる IP プロトコルとポート番号も定義します。Redis の場合、デフォルトのポート番号は 6379 です。最後に、この例は ElastiCache クラスター用のサブネットグループを作成します。このリソースの名前を定義し、そのタイプを宣言して、サブネットグループのサブネットの説明と ID を追加します。

> **Note**
>
> ElastiCache クラスターにはプライベートサブネットを使用することが推奨されます。プライベートサブネットを使用する VPC の詳細については、「https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Scenario2.html」を参照してください。

```
#This sample requires you to create a separate configuration file that defines the
 custom option settings for CacheCluster properties.

Resources:
  MyElastiCache:
    Type: "AWS::ElastiCache::CacheCluster"
    Properties:
      CacheNodeType:
```

```
        Fn::GetOptionSetting:
          OptionName : "CacheNodeType"
          DefaultValue : "cache.t2.micro"
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName : "NumCacheNodes"
          DefaultValue : "1"
      Engine:
        Fn::GetOptionSetting:
          OptionName : "Engine"
          DefaultValue : "redis"
      CacheSubnetGroupName:
        Ref: "MyCacheSubnets"
      VpcSecurityGroupIds:
        - Ref: "MyCacheSecurityGroup"
  MyCacheSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: "Lock cache down to webserver access only"
      VpcId:
        Fn::GetOptionSetting:
          OptionName : "VpcId"
      SecurityGroupIngress :
        - IpProtocol : "tcp"
          FromPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          ToPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          SourceSecurityGroupId:
            Ref: "AWSEBSecurityGroup"
  MyCacheSubnets:
    Type: "AWS::ElastiCache::SubnetGroup"
    Properties:
      Description: "Subnets for ElastiCache"
      SubnetIds:
        Fn::GetOptionSetting:
          OptionName : "CacheSubnets"
Outputs:
  ElastiCache:
    Description : "ID of ElastiCache Cache Cluster with Redis Engine"
```

```
    Value :
      Ref : "MyElastiCache"
```

この設定ファイル例で使用されているリソースの詳細については、以下を参照してください。

- AWS::ElastiCache::CacheCluster
- AWS::EC2::SecurityGroup
- AWS::ElastiCache::SubnetGroup

次に、オプションの設定ファイルに options.config という名前を付け、カスタムオプションの設定を定義します。

> **Note**
>
> 以下の例では、例の CacheSubnets および VpcId の値をユーザー独自のサブネットおよび VPC に置き換えます。

```
option_settings:
  "aws:elasticbeanstalk:customoption":
    CacheNodeType : cache.t2.micro
    NumCacheNodes : 1
    Engine : redis
    CachePort : 6379
    CacheSubnets:
      - subnet-1a1a1a1a
      - subnet-2b2b2b2b
      - subnet-3c3c3c3c
    VpcId: vpc-4d4d4d4d
```

このコードは、設定ファイル (この例では options.config) の CacheNodeType、NumCacheNodes、Engine、CachePort、CacheSubnets、および VpcId の値から CacheNodeType、NumCacheNodes、Engine、CachePort、CacheSubnets、および VpcId プロパティの値を取得するように Elastic Beanstalk に指示します。そのファイルには aws:elasticbeanstalk:customoption セクションが含まれており (option_settings に)、そこにはサンプルの値を示す名前と値のペアが含まれています。上の例では、cache.t2.micro、1、redis、6379、subnet-1a1a1a1a、subnet-2b2b2b2b、subnet-3c3c3c

および vpc-4d4d4d4d が値として使用されます。Fn::GetOptionSetting の詳細については、「関数」を参照してください。

例: SQS、CloudWatch、SNS

この例では、Amazon SQS キューおよびキューの深さに対するアラームを、環境に追加します。この例で示されているプロパティは、これらの各リソースに対して設定する必要のある最低限必要なプロパティです。サンプルは、「SQS, SNS, and CloudWatch」でダウンロードできます。

> **Note**
>
> この例では、課金される可能性のある AWS リソースを作成します。 AWS 料金の詳細については、「」を参照してくださいhttps://aws.amazon.com/pricing/。一部の サービスは AWS 無料利用枠の一部です。新規のお客様は、無料でこれらのサービスをテストできる場合があります。詳細については「https://aws.amazon.com/free/」を参照してください。

この例を使用するには、以下を実行します。

1. ソースバンドルの最上位ディレクトリに .ebextensions ディレクトリを作成します。
2. 拡張子が .config の設定ファイルを 2 つ作成し、.ebextensions ディレクトリに配置します。一方の設定ファイルではリソースが定義され、もう一方の設定ファイルではオプションが定義されます。
3. アプリケーションを Elastic Beanstalk にデプロイします。

   YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

リソースを定義する設定ファイル (例: sqs.config) を作成します。この例では、まず SQS キューを作成し、次に VisbilityTimeout リソースに MySQSQueue プロパティを定義します。次に、SNS Topic を作成し、アラームが発生したら someone@example.com にメールを送信するように指定します。最後に、キューのメッセージが 10 を超えたら CloudWatch アラームを作成します。Dimensions プロパティでは、ディメンションの名前およびディメンションの測定を表す値を指定します。Fn::GetAtt を使用して、QueueName から MySQSQueue の値を返します。

```
#This sample requires you to create a separate configuration file to define the custom
 options for the SNS topic and SQS queue.
```

```
Resources:
  MySQSQueue:
    Type: AWS::SQS::Queue
    Properties:
      VisibilityTimeout:
        Fn::GetOptionSetting:
          OptionName: VisibilityTimeout
          DefaultValue: 30
  AlarmTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint:
            Fn::GetOptionSetting:
              OptionName: AlarmEmail
              DefaultValue: "nobody@amazon.com"
          Protocol: email
  QueueDepthAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: "Alarm if queue depth grows beyond 10 messages"
      Namespace: "AWS/SQS"
      MetricName: ApproximateNumberOfMessagesVisible
      Dimensions:
        - Name: QueueName
          Value : { "Fn::GetAtt" : [ "MySQSQueue", "QueueName"] }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 1
      Threshold: 10
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: AlarmTopic
      InsufficientDataActions:
        - Ref: AlarmTopic

Outputs :
  QueueURL:
    Description : "URL of newly created SQS Queue"
    Value : { Ref : "MySQSQueue" }
  QueueARN :
    Description : "ARN of newly created SQS Queue"
    Value : { "Fn::GetAtt" : [ "MySQSQueue", "Arn"]}
  QueueName :
```

```
    Description : "Name newly created SQS Queue"
    Value : { "Fn::GetAtt" : [ "MySQSQueue", "QueueName"]}
```

この設定ファイル例で使用されているリソースの詳細については、以下を参照してください。

- AWS::SQS::Queue
- AWS::SNS::Topic
- AWS::CloudWatch::Alarm

options.config という名前の別の設定ファイルを作成し、カスタムオプションの設定を定義します。

```
option_settings:
  "aws:elasticbeanstalk:customoption":
     VisibilityTimeout : 30
     AlarmEmail : "nobody@example.com"
```

これらの行は、Elastic Beanstalk に対し、VisibilityTimeout および Subscription Endpoint プロパティの値を、option_settings セクションと、使用する実際の値を含む名前と値のペアを含む aws:elasticbeanstalk:customoption セクションが含まれている設定ファイル (この例では options.config) の VisibilityTimeout および Subscription Endpoint の値から取得するように指示します。上の例では、これは値として 30 および "nobody@amazon.com" を使用することを意味します。Fn::GetOptionSetting の詳細については、「the section called “関数”」を参照してください。

例: DynamoDB、CloudWatch、SNS

この設定ファイルは、 AWS SDK for PHP 2 を使用して、PHP ベースのアプリケーションのセッションハンドラーとして DynamoDB テーブルを設定します。この例を使用するには、IAM インスタンスプロファイルが必要です。これは、お客様の環境内のインスタンスに追加され、DynamoDB テーブルにアクセスするために使用されます。

このステップで使用するサンプルは、「DynamoDB session Support example」からダウンロードできます。サンプルには以下のファイルが含まれています。

- サンプルアプリケーション、index.php
- DynamoDB テーブルおよびその他の AWS リソースを作成および設定しdynamodb.config、Elastic Beanstalk 環境でアプリケーションをホストする EC2 インスタンスにソフトウェアをインストールする設定ファイル 。

- dynamodb.config のデフォルト設定をこの特定のインストールに固有の設定で上書きする、設定ファイル options.config

index.php

```
<?php

// Include the SDK using the Composer autoloader
require '../vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;

// Grab the session table name and region from the configuration file
list($tableName, $region) = file(__DIR__ . '/../sessiontable');
$tableName = rtrim($tableName);
$region = rtrim($region);

// Create a DynamoDB client and register the table as the session handler
$dynamodb = DynamoDbClient::factory(array('region' => $region));
$handler = $dynamodb->registerSessionHandler(array('table_name' => $tableName,
 'hash_key' => 'username'));

// Grab the instance ID so we can display the EC2 instance that services the request
$instanceId = file_get_contents("http://169.254.169.254/latest/meta-data/instance-id");
?>
<h1>Elastic Beanstalk PHP Sessions Sample</h1>
<p>This sample application shows the integration of the Elastic Beanstalk PHP
container and the session support for DynamoDB from the AWS SDK for PHP 2.
Using DynamoDB session support, the application can be scaled out across
multiple web servers. For more details, see the
<a href="https://aws.amazon.com/php/">PHP Developer Center</a>.</p>

<form id="SimpleForm" name="SimpleForm" method="post" action="index.php">
<?php
echo 'Request serviced from instance ' . $instanceId . '<br/>';
echo '<br/>';

if (isset($_POST['continue'])) {
  session_start();
  $_SESSION['visits'] = $_SESSION['visits'] + 1;
  echo 'Welcome back ' . $_SESSION['username'] . '<br/>';
  echo 'This is visit number ' . $_SESSION['visits'] . '<br/>';
  session_write_close();
```

```
  echo '<br/>';
  echo '<input type="Submit" value="Refresh" name="continue" id="continue"/>';
  echo '<input type="Submit" value="Delete Session" name="killsession"
 id="killsession"/>';
} elseif (isset($_POST['killsession'])) {
  session_start();
  echo 'Goodbye ' . $_SESSION['username'] . '<br/>';
  session_destroy();
  echo 'Username: <input type="text" name="username" id="username" size="30"/><br/>';
  echo '<br/>';
  echo '<input type="Submit" value="New Session" name="newsession" id="newsession"/>';
} elseif (isset($_POST['newsession'])) {
  session_start();
  $_SESSION['username'] = $_POST['username'];
  $_SESSION['visits'] = 1;
  echo 'Welcome to a new session ' . $_SESSION['username'] . '<br/>';
  session_write_close();
  echo '<br/>';
  echo '<input type="Submit" value="Refresh" name="continue" id="continue"/>';
  echo '<input type="Submit" value="Delete Session" name="killsession"
 id="killsession"/>';
} else {
  echo 'To get started, enter a username.<br/>';
  echo '<br/>';
  echo 'Username: <input type="text" name="username" id="username" size="30"/><br/>';
  echo '<input type="Submit" value="New Session" name="newsession" id="newsession"/>';
}
?>
</form>
```

.ebextensions/dynamodb.config

```
Resources:
  SessionTable:
    Type: AWS::DynamoDB::Table
    Properties:
      KeySchema:
        HashKeyElement:
          AttributeName:
            Fn::GetOptionSetting:
              OptionName : SessionHashKeyName
              DefaultValue: "username"
          AttributeType:
```

```
            Fn::GetOptionSetting:
              OptionName : SessionHashKeyType
              DefaultValue: "S"
      ProvisionedThroughput:
        ReadCapacityUnits:
          Fn::GetOptionSetting:
            OptionName : SessionReadCapacityUnits
            DefaultValue: 1
        WriteCapacityUnits:
          Fn::GetOptionSetting:
            OptionName : SessionWriteCapacityUnits
            DefaultValue: 1

  SessionWriteCapacityUnitsLimit:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: { "Fn::Join" : ["", [{ "Ref" : "AWSEBEnvironmentName" }, "
 write capacity limit on the session table." ]]}
      Namespace: "AWS/DynamoDB"
      MetricName: ConsumedWriteCapacityUnits
      Dimensions:
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 12
      Threshold:
          Fn::GetOptionSetting:
            OptionName : SessionWriteCapacityUnitsAlarmThreshold
            DefaultValue: 240
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: SessionAlarmTopic
      InsufficientDataActions:
        - Ref: SessionAlarmTopic

  SessionReadCapacityUnitsLimit:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: { "Fn::Join" : ["", [{ "Ref" : "AWSEBEnvironmentName" }, " read
 capacity limit on the session table." ]]}
      Namespace: "AWS/DynamoDB"
      MetricName: ConsumedReadCapacityUnits
      Dimensions:
```

```
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 12
      Threshold:
          Fn::GetOptionSetting:
            OptionName : SessionReadCapacityUnitsAlarmThreshold
            DefaultValue: 240
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: SessionAlarmTopic
      InsufficientDataActions:
        - Ref: SessionAlarmTopic

  SessionThrottledRequestsAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: { "Fn::Join" : ["", [{ "Ref" : "AWSEBEnvironmentName" }, ":
 requests are being throttled." ]]}
      Namespace: AWS/DynamoDB
      MetricName: ThrottledRequests
      Dimensions:
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 1
      Threshold:
        Fn::GetOptionSetting:
          OptionName: SessionThrottledRequestsThreshold
          DefaultValue: 1
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: SessionAlarmTopic
      InsufficientDataActions:
        - Ref: SessionAlarmTopic

  SessionAlarmTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint:
            Fn::GetOptionSetting:
```

```
                OptionName: SessionAlarmEmail
                DefaultValue: "nobody@amazon.com"
            Protocol: email

files:
  "/var/app/sessiontable":
    mode: "000444"
    content: |
      `{"Ref" : "SessionTable"}`
      `{"Ref" : "AWS::Region"}`

  "/var/app/composer.json":
    mode: "000744"
    content:
      {
        "require": {
           "aws/aws-sdk-php": "*"
        }
      }

container_commands:
 "1-install-composer":
   command: "cd /var/app; curl -s http://getcomposer.org/installer | php"
 "2-install-dependencies":
   command: "cd /var/app; php composer.phar install"
 "3-cleanup-composer":
   command: "rm -Rf /var/app/composer.*"
```

サンプルの設定ファイルでは、最初に DynamoDB テーブルを作成して、主キー構造と、十分なリソースを割り当てて必要なスループットを提供するためのキャパシティーユニットを設定します。次に、`WriteCapacity` および `ReadCapacity` に対する CloudWatch アラームを作成します。アラームしきい値を超過した場合にメールを "nobody@amazon.com" に送信する SNS トピックを作成します。

環境の AWS リソースを作成して設定したら、EC2 インスタンスをカスタマイズする必要があります。`files` キーを使用して、DynamoDB テーブルの詳細を環境の EC2 インスタンスに渡し、 AWS SDK for PHP 2 の `composer.json`ファイルに「必須」を追加します。最後に、コンテナコマンドを実行して、コンポーザーおよび必要な依存関係をインストールし、インストーラを削除します。

`.ebextensions/options.config`

```
option_settings:
```

```
  "aws:elasticbeanstalk:customoption":
     SessionHashKeyName                      : username
     SessionHashKeyType                      : S
     SessionReadCapacityUnits                : 1
     SessionReadCapacityUnitsAlarmThreshold  : 240
     SessionWriteCapacityUnits               : 1
     SessionWriteCapacityUnitsAlarmThreshold : 240
     SessionThrottledRequestsThreshold       : 1
     SessionAlarmEmail                       : me@example.com
```

SessionAlarmEmail 値をアラーム通知用の E メールに置き換えます。`options.config` ファイルには、`dynamodb.config` で定義されている変数の一部に使用する値が含まれます。たとえば、`dynamodb.config` には以下のような行があります。

```
Subscription:
  - Endpoint:
      Fn::GetOptionSetting:
        OptionName: SessionAlarmEmail
        DefaultValue: "nobody@amazon.com"
```

これらの行は、Elastic Beanstalk に対し、Endpoint プロパティの値を、option_settings セクションと、使用する実際の値を含む名前と値のペアを含む aws:elasticbeanstalk:customoption セクションが含まれている設定ファイル (このサンプルアプリケーションでは `options.config`) の SessionAlarmEmail の値から取得するように指示します。上の例では、これは SessionAlarmEmail に値 `nobody@amazon.com` が割り当てられることを意味します。

この例で使用されている CloudFormation リソースの詳細については、以下を参照してください。

- AWS::DynamoDB::Table
- AWS::CloudWatch::Alarm
- AWS::SNS::Topic

Elastic Beanstalk の保存された設定を使用する

環境の作成中に他の環境または実行中の環境に適用できる環境設定を、オブジェクトとして Amazon Simple Storage Service (Amazon S3) に保存することができます。保存された設定は、YAML 形式のテンプレートであり、環境の プラットフォームバージョン、階層、設定オプションの設定、およびタグを定義します。

保存された設定の作成時にタグを適用したり、既存の保存された設定のタグを編集したりできます。保存された設定に適用したタグは、`Tags:` キーを使用して保存された設定に指定したタグとは関係ありません。後者のタグは、保存された設定を環境に適用するときに、環境に適用されます。詳細については、「保存された設定にタグ付けする」を参照してください。

> **Note**
>
> いくつかの方法を使用して、保存した設定を作成して Elastic Beanstalk 環境に適用できます。これらには、Elastic Beanstalk コンソール、EB CLI、および AWS CLIが含まれます。保存された設定を作成および適用する他の方法の例については、次のトピックを参照してください。
>
> - 環境を作成する前に設定オプションを設定する
> - 環境の作成時の設定オプションの設定
> - 環境の作成後に設定オプションを設定する

Elastic Beanstalk マネジメントコンソールで環境の現在の状態から、保存された設定を作成します。

環境の設定を保存するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Save Configuration] (設定の保存) の順に選択します。
4. 画面上のフォームを使用して、保存した設定に名前を付けます。必要に応じて、簡単な説明を入力し、タグのキーと値を追加します。
5. [保存] を選択します。

保存された設定には、コンソールまたは Elastic Beanstalk API を使用する他のクライアントで環境に適用したその他の設定が含まれます。次に、保存された設定を後で環境に適用して以前の状態に戻したり、環境の作成時に新しい環境に適用したりできます。

次の例に示すように、EB CLI の the section called “eb config” コマンドを使用して設定をダウンロードできます。*NAME* は保存された設定の名前です。

```
eb config get NAME
```

環境の作成時に保存された設定を適用するには (Elastic Beanstalk コンソール)

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. ナビゲーションペインで、アプリケーションの名前を見つけ、[保存された設定] を選択します。
4. 適用する保存済み設定を選択し、[Launch environment (環境の起動)] を選択します。
5. ウィザードを続行して環境を作成します。

保存された設定には、アプリケーションのソースコードの設定ファイルを使用して適用した設定は含まれません。同じ設定が設定ファイルと保存された設定の両方に適用されている場合、保存された設定が優先されます。同様に、Elastic Beanstalk コンソールで指定されるオプションは保存されている設定を上書きします。詳細については、「優先順位」を参照してください。

保存された設定は、アプリケーションに関連する名前のフォルダ内の Elastic Beanstalk S3 バケットに格納されます。例えば、us-west-2 リージョンにあるアカウント番号 123456789012 の `my-app` という名前のアプリケーションの設定は、`s3://elasticbeanstalk-us-west-2-123456789012/resources/templates/my-app/` にあります。

保存された設定をテキストエディタで開いて内容を表示します。次の例は、Elastic Beanstalk マネジメントコンソールで起動されたウェブサーバーの環境の設定を示します。

```
EnvironmentConfigurationMetadata:
  Description: Saved configuration from a multicontainer Docker environment created
 with the Elastic Beanstalk Management Console
  DateCreated: '1520633151000'
  DateModified: '1520633151000'
Platform:
  PlatformArn: arn:aws:elasticbeanstalk:us-east-2::platform/Java 8 running on 64bit
 Amazon Linux/2.5.0
OptionSettings:
  aws:elasticbeanstalk:command:
    BatchSize: '30'
    BatchSizeType: Percentage
  aws:elasticbeanstalk:sns:topics:
    Notification Endpoint: me@example.com
  aws:elb:policies:
    ConnectionDrainingEnabled: true
    ConnectionDrainingTimeout: '20'
```

```
  aws:elb:loadbalancer:
    CrossZone: true
  aws:elasticbeanstalk:environment:
    ServiceRole: aws-elasticbeanstalk-service-role
  aws:elasticbeanstalk:application:
    Application Healthcheck URL: /
  aws:elasticbeanstalk:healthreporting:system:
    SystemType: enhanced
  aws:autoscaling:launchconfiguration:
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
    InstanceType: t2.micro
    EC2KeyName: workstation-uswest2
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Health
    RollingUpdateEnabled: true
EnvironmentTier:
  Type: Standard
  Name: WebServer
AWSConfigurationTemplateVersion: 1.1.0.0
Tags:
  Cost Center: WebApp Dev
```

保存された設定の内容を変更し、Amazon S3 の同じ場所にデータを保存することができます。適切な場所に適切な形式で保存された設定は、Elastic Beanstalk マネジメントコンソールを使用して環境に適用できます。

以下のキーがサポートされます。

- AWSConfigurationTemplateVersion (必須) – 設定テンプレートバージョン (1.1.0.0)。

```
AWSConfigurationTemplateVersion: 1.1.0.0
```

- Platform – 環境のプラットフォームバージョンの Amazon リソースネーム (ARN)。プラットフォームは、ARN またはソリューションスタック名で指定できます。

```
Platform:
  PlatformArn: arn:aws:elasticbeanstalk:us-east-2::platform/Java 8 running on 64bit
 Amazon Linux/2.5.0
```

- SolutionStack – 環境を作成するために使用されるソリューションスタックの完全名。

```
SolutionStack: 64bit Amazon Linux 2017.03 v2.5.0 running Java 8
```

- OptionSettings – 環境に適用する設定オプションの設定。たとえば、次のエントリはインスタンスタイプを t2.micro に設定します。

```
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: t2.micro
```

- Tags – 環境内で作成されたリソースに適用される最大 47 個のタグ。

```
Tags:
  Cost Center: WebApp Dev
```

- EnvironmentTier – 作成する環境のタイプ。ウェブサーバーの環境では、このセクションは除外できます (ウェブサーバーがデフォルトとなります)。ワーカー環境では、次を使用します。

```
EnvironmentTier:
  Name: Worker
  Type: SQS/HTTP
```

> **Note**
>
> いくつかの方法を使用して、保存した設定を作成して Elastic Beanstalk 環境に適用できます。これらには、Elastic Beanstalk コンソール、EB CLI、および AWS CLIが含まれます。保存された設定を作成および適用する他の方法の例については、次のトピックを参照してください。
>
> - 環境を作成する前に設定オプションを設定する
> - 環境の作成時の設定オプションの設定
> - 環境の作成後に設定オプションを設定する

保存された設定にタグ付けする

AWS Elastic Beanstalk 保存した設定にタグを適用できます。タグは、 AWS リソースに関連付けられたキーと値のペアです。Elastic Beanstalk リソースのタグ付け、ユースケース、タグのキーと値の制約、サポートされているリソースタイプの詳細については、「Elastic Beanstalk アプリケーションリソースのタグ付け」を参照してください。

保存された設定を作成するときにタグを指定できます。既存の保存された設定では、タグの追加や削除、既存タグの値の更新ができます。保存された設定ごとに最大 50 個のタグを追加できます。

保存された設定の作成時にタグを追加する

Elastic Beanstalk コンソールを使用して設定を保存するときに、[設定の保存] ページでタグのキーと値を指定できます。

EB CLI を使用して設定を保存した場合は、eb config で `--tags` オプションを使用してタグを追加します。

```
~/workspace/my-app$ eb config --tags mytag1=value1,mytag2=value2
```

AWS CLI または他の API ベースのクライアントで、 create-configuration-template コマンドの `--tags`パラメータを使用してタグを追加します。

```
$ aws elasticbeanstalk create-configuration-template \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --application-name my-app --template-name my-template --solution-stack-
name solution-stack
```

既存の保存された設定のタグを管理する

Elastic Beanstalk の既存の保存された設定のタグを追加、更新、および削除できます。

Elastic Beanstalk コンソールを使用して保存された設定のタグを管理するには

1. Elastic Beanstalk 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、[アプリケーション] を選択し、リストからアプリケーションの名前を選択します。
3. ナビゲーションペインで、アプリケーションの名前を見つけ、[保存された設定] を選択します。
4. 管理する保存済み設定を選択します。
5. [Actions (アクション)] を選択して [Manage tags (タグの管理)] を選択します。
6. 画面上のフォームを使用して、タグを追加、更新、または削除します。
7. ページの最下部で [適用] を選択し変更を保存します。

EB CLI を使用して保存された設定を更新する場合は、eb tags を使用してタグを追加、更新、削除、一覧表示します。

たとえば、次のコマンドでは、保存された設定のタグを一覧表示します。

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:configurationtemplate/my-app/my-template"
```

次のコマンドでは、mytag1 タグを更新して mytag2 タグを削除します。

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:configurationtemplate/my-app/my-template"
```

オプションの完全なリストおよび詳細な例については、「eb tags」を参照してください。

AWS CLI または他の API ベースのクライアントでは、 list-tags-for-resource コマンドを使用して、保存された設定のタグを一覧表示します。

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:configurationtemplate/my-app/my-template"
```

保存された設定のタグを追加、更新、または削除するには、 update-tags-for-resource コマンドを使用します。

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:configurationtemplate/my-app/my-template"
```

追加するタグと更新するタグを update-tags-for-resource の `--tags-to-add` パラメータで指定します。存在していないタグが追加され、既存のタグの値が更新されます。

> **Note**
>
> Elastic Beanstalk で保存された設定で一部の EB CLI および AWS CLI コマンドを使用するには、保存された設定の ARN が必要です。ARN を作成するには、最初に次のコマンドを使用して保存された設定の名前を取得します。
>
> ```
> $ aws elasticbeanstalk describe-applications --application-names my-app
> ```

コマンドの出力で ConfigurationTemplates キーを探します。この要素は、保存された設定の名前を示します。この名前を、このページに記載されているコマンドの *my-template* が指定されている場所で使用します。

マニフェスト環境 (env.yaml)

環境を作成するときに使用する、環境の名前、ソリューションスタックと環境リンクを設定するために、YAML 形式の環境マニフェストをアプリケーションソースバンドルのルートに含めることができます。

このファイル形式には環境グループのサポートが含まれます。グループを使用するには、マニフェスト内の環境の名前の末尾に +記号を指定します。環境を作成あるいは更新する場合、グループ名を `--group-name` (AWS CLI) または `--env-group-suffix` (EB CLI) と指定します。グループの詳細については、「Elastic Beanstalk 環境のグループを作成および更新する」を参照してください。

次の例では、マニフェストが、ウェブサーバー環境に依存しているワーカー環境のコンポーネントへのリンクを持つウェブサーバー環境を定義します。マニフェストは、グループを使用して同じソースバンドルで複数の環境を作成することを許可します。

~/myapp/frontend/env.yaml

```
AWSConfigurationTemplateVersion: 1.1.0.0
SolutionStack: 64bit Amazon Linux 2015.09 v2.0.6 running Multi-container Docker 1.7.1
 (Generic)
OptionSettings:
  aws:elasticbeanstalk:command:
    BatchSize: '30'
    BatchSizeType: Percentage
  aws:elasticbeanstalk:sns:topics:
    Notification Endpoint: me@example.com
  aws:elb:policies:
    ConnectionDrainingEnabled: true
    ConnectionDrainingTimeout: '20'
  aws:elb:loadbalancer:
    CrossZone: true
  aws:elasticbeanstalk:environment:
    ServiceRole: aws-elasticbeanstalk-service-role
  aws:elasticbeanstalk:application:
    Application Healthcheck URL: /
  aws:elasticbeanstalk:healthreporting:system:
```

```
    SystemType: enhanced
  aws:autoscaling:launchconfiguration:
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
    InstanceType: t2.micro
    EC2KeyName: workstation-uswest2
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Health
    RollingUpdateEnabled: true
Tags:
  Cost Center: WebApp Dev
CName: front-A08G28LG+
EnvironmentName: front+
EnvironmentLinks:
  "WORKERQUEUE" : "worker+"
```

以下のキーがサポートされます。

- AWSConfigurationTemplateVersion (必須) – 設定テンプレートバージョン (1.1.0.0)。

```
AWSConfigurationTemplateVersion: 1.1.0.0
```

- Platform – 環境のプラットフォームバージョンの Amazon リソースネーム (ARN)。プラットフォームは、ARN またはソリューションスタック名で指定できます。

```
Platform:
  PlatformArn: arn:aws:elasticbeanstalk:us-east-2::platform/Java 8 running on 64bit
 Amazon Linux/2.5.0
```

- SolutionStack – 環境を作成するために使用されるソリューションスタックの完全名。

```
SolutionStack: 64bit Amazon Linux 2017.03 v2.5.0 running Java 8
```

- OptionSettings – 環境に適用する設定オプションの設定。たとえば、次のエントリはインスタンスタイプを t2.micro に設定します。

```
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: t2.micro
```

- Tags – 環境内で作成されたリソースに適用される最大 47 個のタグ。

```
Tags:
```

```
  Cost Center: WebApp Dev
```

- EnvironmentTier – 作成する環境のタイプ。ウェブサーバーの環境では、このセクションは除外できます (ウェブサーバーがデフォルトとなります)。ワーカー環境では、次を使用します。

```
EnvironmentTier:
  Name: Worker
  Type: SQS/HTTP
```

- CName – 環境用の CNAME。グループを有効にするために、名前の末尾に + 記号を含めます。

```
CName: front-A08G28LG+
```

- EnvironmentName – 作成する環境の名前。グループを有効にするために、名前の末尾に + 記号を含めます。

```
EnvironmentName: front+
```

  グループが有効化されている場合、環境を作成するときにグループ名を指定する必要があります。Elastic Beanstalk は、環境の名前にハイフンでグループ名を追加します。たとえば、環境の名前が `front+` で、グループ名が `dev` の場合、Elastic Beanstalk は環境の作成時に名前を `front-dev` とします。

- EnvironmentLinks – 依存関係にある変数名と環境の名前のマップ。次の例では、`worker+` 環境を依存化し、`WORKERQUEUE` という変数にリンク情報を保存するように Elastic Beanstalk に指示しています。

```
EnvironmentLinks:
  "WORKERQUEUE" : "worker+"
```

  リンク変数の値は、リンクされた環境の種類によって異なります。ウェブサーバー環境の場合、リンクは環境の CNAME となります。ワーカーの環境では、リンクは環境内の Amazon Simple Queue Service (Amazon SQS) キューの名前となります。

[CName]、[EnvironmentName]、[EnvironmentLinks] キーは、環境グループや他の環境へのリンクを作成するのに使用できます。これらの機能は、EB CLI AWS CLI または SDK を使用する場合に現在サポートされています。

Elastic Beanstalk 環境でのカスタム Amazon マシンイメージ (AMI) の使用

このセクションでは、カスタム AMI の使用を検討するタイミングと、環境内のカスタム AMI を設定および管理する手順について説明します。 AWS Elastic Beanstalk 環境を作成するときに、プラットフォームバージョンに含まれる標準の Elastic Beanstalk AMI の代わりに使用する Amazon マシンイメージ (AMI) を指定できます。カスタム AMI では、標準 AMI に含まれていない多数のソフトウェアをインストールする必要がある場合、インスタンスがユーザーの環境で起動されていればプロビジョニング時間が向上します。

設定ファイルを使用すると、環境を迅速かつ一貫してカスタマイズできます。ただし、設定を適用すると、環境の作成および更新時に時間がかかるようになることがあります。設定ファイルで多数のサーバーの設定をする場合は、必要なソフトウェアのインストールおよび設定が済んでいるカスタム AMI を作成することによって、その時間を短縮できます。

設定ファイルでは実装が困難であったり適用に時間がかかったりする低レベルのコンポーネント (Linux カーネルなど) に対する変更を、カスタム AMI で行うこともできます。カスタム AMI を作成するには、Elastic Beanstalk プラットフォーム AMI を Amazon EC2 で起動し、ニーズに合わせてソフトウェアと設定をカスタマイズした後、インスタンスを停止して、そこから AMI を保存します。

カスタム AMI の作成

EC2 Image Builder を使用して、これらの手順の代替としてカスタム AMI を作成および管理できます。詳細については、「Image Builder User Guide」を参照してください。

ベースの Elastic Beanstalk AMI を特定するには

1. コマンドウィンドウで、次のようなコマンドを実行します。詳細については、「AWS CLI コマンドリファレンス」の describe-platform-version を参照してください。

   カスタム AMI を使用する AWS リージョンを指定し、プラットフォーム ARN とバージョン番号を、アプリケーションが基づいている Elastic Beanstalk プラットフォームに置き換えます。

   Example - Mac OS / Linux OS

```
$ aws elasticbeanstalk describe-platform-version --region us-east-2 \
      --platform-arn "arn:aws:elasticbeanstalk:us-east-2::platform/Node.js 20
 running on 64bit Amazon Linux 2023/6.1.7" \
```

```
      --query PlatformDescription.CustomAmiList
[
    {
        "VirtualizationType": "pv",
        "ImageId": ""
    },
    {
        "VirtualizationType": "hvm",
        "ImageId": "ami-020ae06fdda6a0f66"
    }
]
```

Example - Windows OS

```
C:\> aws elasticbeanstalk describe-platform-version --region us-east-2 --platform-
arn"arn:aws:elasticbeanstalk:us-east-2::platform/
IIS 10.0 running on 64bit Windows Server 2022/2.15.3" --query
 PlatformDescription.CustomAmiList
[
    {
        "VirtualizationType": "pv",
        "ImageId": ""
    },
    {
        "VirtualizationType": "hvm",
        "ImageId": "ami-020ae06fdda6a0f66"
    }
]
```

2. 結果として `ami-020ae06fdda6a0f66` のようになる `ImageId` 値を書き留めます。

値は、アプリケーションに関連するプラットフォームバージョン、EC2 インスタンスアーキテクチャ、 AWS リージョンのストック Elastic Beanstalk AMI です。複数のプラットフォーム、アーキテクチャ、または AWS リージョンの AMIs を作成する必要がある場合は、このプロセスを繰り返して、組み合わせごとに正しいベース AMI を特定します。

> **Note**
>
> Elastic Beanstalk 環境で起動しているインスタンスから AMI を作成しないでください。プロビジョニング時に Elastic Beanstalk がインスタンスに変更を加えるため、保存された AMI で問題が発生する可能性があります。Elastic Beanstalk 環境のインスタンスからイメージを保

存すると、そのインスタンスにデプロイされたアプリケーションのバージョンがイメージの固定部分になります。

Linux の場合、Elastic Beanstalk が発行したものではないコミュニティ AMI からカスタム AMI を作成することもできます。最新の Amazon Linux AMI を出発点として使用できます。Elastic Beanstalk によって管理されていない Linux AMI を使用して環境を起動すると、Elastic Beanstalk は、拡張ヘルスレポートなどの機能をサポートするために、プラットフォームソフトウェア (言語、フレームワーク、プロキシサーバーなど) と追加のコンポーネントのインストールを試みます。

> **Note**
>
> Windows Server に基づくカスタム AMI には、ステップ 1 で前述したように、`describe-platform-version` から返されたストックの Elastic Beanstalk AMI が必要です。

Elastic Beanstalk は Elastic Beanstalk によって管理されていない AMI を使用できますが、Elastic Beanstalk が欠落したコンポーネントをインストールするためにプロビジョニング時間が増加し、そもそもカスタム AMI を作成する利点が少なくなったり、まったくなくなったりします。他の Linux ディストリビューションは一部のトラブルシューティングについては動作しますが、公式にはサポートされていません。アプリケーションに特定の Linux ディストリビューションが必要な場合は、代わりの方法として Docker イメージを作成し、Elastic Beanstalk の Docker プラットフォームまたはマルチコンテナ Docker プラットフォームで実行することもできます。

カスタム AMI を作成するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. [インスタンスの作成] を選択してください。
3. ベースの Elastic Beanstalk AMI (`describe-platform-version` を使用) または Amazon Linux AMI を特定したら、その AMI ID を検索ボックスに入力します。次に、Enter キーを押します。

   ニーズに合った別のコミュニティ AMI のリストを検索することもできます。

   > **Note**
   >
   > HVM 仮想化を使用する AMI を選択することをお勧めします。これらの AMI の説明には、[仮想化タイプ: hvm] と表示されます。

詳細については、「Amazon EC2 ユーザーガイド」の「仮想化タイプ」を参照してください。

4. [選択] で AMI を選択します。
5. インスタンスタイプを選択し、[次: インスタンスの詳細の設定] を選択します。
6. (廃止された Amazon Linux AMI (AL1) プラットフォームの場合) サポートされている Linux ベースのプラットフォームまたは Windows プラットフォームで環境が実行されている場合は、このステップをスキップします。

   [高度な詳細] セクションを展開し、[ユーザーデータ] フィールドに以下のテキストを貼り付けます。

   ```
   #cloud-config
     repo_releasever: repository version number
     repo_upgrade: none
   ```

   リポジトリバージョン番号は、AMI 名の年と月のバージョンです。例えば、2015 年 3 月リリースの Amazon Linux に基づく AMI のリポジトリバージョン番号は `2015.03` です。Elastic Beanstalk イメージの場合、この値は、Amazon Linux AMI (Amazon Linux 2 より前) に基づくプラットフォームバージョンのソリューションスタック名に示される日付と同じです。

   > **Note**
   >
   > `repo_releasever` 設定では、Amazon Linux AMI のロックオン起動機能を指定します。これにより、AMI は起動時に固定された特定のリポジトリバージョンを使用します。この機能は Amazon Linux 2 ではサポートされないため、環境で最新の Amazon Linux 2 プラットフォームブランチが使用されている場合は、指定しないでください。カスタム AMI を Elastic Beanstalk で使用する場合は、Amazon Linux AMI プラットフォームブランチ (Amazon Linux 2 より前) でのみ、この設定が必要です。
   > `repo_upgrade` を設定すると、セキュリティ更新プログラムの自動インストールが無効になります。これは、Elastic Beanstalk でカスタム AMI を使用するために必要となります。

7. ウィザードを続行して、EC2 インスタンスを起動します。プロンプトが表示されたら、次のステップでそのインスタンスに接続できるように、アクセス権があるキーペアを選択します。
8. SSH または RDP でインスタンスに接続します。
9. 目的のカスタマイズを行います。

10. (Windows プラットフォーム) EC2Config サービス Sysprep を実行します。EC2Config の詳細については、「EC2Config サービスを使用した Windows インスタンスの設定」を参照してください。 AWS Management Consoleから取得できるランダムなパスワードを生成するように Sysprep が設定されていることを確認します。
11. Amazon EC2 コンソールで、EC2 インスタンスを停止します。次に、[インスタンスの操作] メニューで、[イメージの作成 (EBS AMI)] を選択します。
12. AWS 追加料金が発生しないようにするには、EC2 インスタンスを終了します。

Elastic Beanstalk 環境でカスタム AMI を使用するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [容量] 設定カテゴリで、[編集] を選択します。
5. [AMI ID] には、ユーザーのカスタム AMI ID を入力します。
6. ページの最下部で [適用] を選択し変更を保存します。

カスタム AMI で新しい環境を作成するときは、AMI の作成時にベースとして使用したのと同じプラットフォームバージョンを使用する必要があります。

カスタム AMI を使用した環境の管理

プラットフォームの更新

カスタム AMI を使用する場合、Elastic Beanstalk は、更新が手動で適用されるか、マネージドプラットフォーム更新を介して適用されるかにかかわらず、プラットフォームバージョンが更新されても、環境内で同じカスタム AMI を引き続き使用します。新しいプラットフォームバージョンのストック AMI を使用するように環境はリセットされません。

新しいプラットフォームバージョンのストック AMI に基づいて、新しいカスタム AMI を作成することをお勧めします。これにより、新しいプラットフォームバージョンで利用可能なパッチが適用され、互換性のないパッケージまたはライブラリバージョンによるデプロイの失敗も最小限に抑えられます。

新しいカスタム AMI 作成の詳細については、このトピックで前述した「カスタム AMI の作成」を参照してください。

カスタム AMI の削除

環境からカスタム AMI を削除し、環境のプラットフォームバージョンにストック AMI を使用するようにリセットする場合は、次の CLI コマンドを使用します。

```
aws elasticbeanstalk update-environment \
  --application-name my-application \
  --environment-name my-environment \
  --region us-east-1 \
  --options-to-remove Namespace=aws:autoscaling:launchconfiguration,OptionName=ImageId
```

> **Note**
>
> サービスの中断を回避するには、この変更を本稼働環境に適用する前に、アプリケーションをストック AMI でテストします。

カスタム AMI をクリーンアップする

カスタム AMI を終了し、Elastic Beanstalk 環境をもう起動する必要がない場合は、ストレージコストを最小限に抑えるために、AMI をクリーンアップすることを検討してください。カスタム AMI をクリーンアップするには、Amazon EC2 から登録を解除し、関連する他のリソースを削除する必要があります。詳細については、「Linux AMI の登録解除」または「Windows AMI の登録解除」を参照してください。

廃止されたプラットフォームの Amazon マシンイメージ (AMI) へのアクセスを維持する

Elastic Beanstalk は、ブランチが使用するオペレーティングシステムまたは主要コンポーネントのサポートが終了になると、プラットフォームブランチのステータスを「廃止」に設定します。プラットフォームブランチの「ベース」 Elastic Beanstalk AMI をプライベートにして、この古い AMI の使用を禁止することもできます。プライベートにした AAMI を使用する環境ではインスタンスを起動できなくなります。

廃止される前にサポート対象の環境にアプリケーションを移行できない場合、ご使用の環境で、このような状況が発生するになる可能性があります。ベース Elastic Beanstalk AMI がプライベートにされた Beanstalk プラットフォームブランチの環境を更新する必要が生じる場合があります。別のアプローチもあります。環境で使用されているベース Elastic Beanstalk AMI の「コピー」に基づいて既存の環境を更新できます。

このトピックでは、環境で使用されているベース Elastic Beanstalk AMI の「コピー」に基づいて既存の環境を更新するステップとスタンドアロンスクリプトについて説明します。アプリケーションをサポート対象のプラットフォームに移行できたら、引き続きアプリケーションとサポート対象環境を維持するための標準の手順を使用できます。

手動ステップ

ベース Elastic Beanstalk AMI の AMI コピーに基づいて環境を更新するには

1. 環境で使用している AMI を特定します。以下のコマンドは、パラメータで指定した Elastic Beanstalk 環境で使用されている AMI を返します。戻り値は次のステップで source-ami-id として使用します。

   コマンドウィンドウで、次のようなコマンドを実行します。詳細については、「AWS CLI コマンドリファレンス」の「describe-configuration-settings」を参照してください。

   コピーするソース AMI を保存する AWS リージョンを指定します。アプリケーション名と環境名をソース AMI に基づくもので置き換えます。次に示すように、クエリパラメータのテキストを入力します。

   Example

   ```
   >aws elasticbeanstalk describe-configuration-settings \
     --application-name my-application \
     --environment-name my-environment \
     --region us-east-2 \
     --query "ConfigurationSettings[0].OptionSettings[?OptionName=='ImageId'] |
    [0].Value"
   ```

2. AMI をアカウントにコピーします。このコマンドは、前のステップで返された source-ami-id をコピーした結果となる新しい AMI を返します。

   > **Note**
   >
   > このコマンドで出力される新しい AMI ID をメモしておいてください。これは、サンプルコマンドの copied-ami-id の部分を置き換えて口のステップで入力する必要があります。

コマンドウィンドウで、次のようなコマンドを実行します。詳細については、「AWS CLI コマンドリファレンス」の「copy-image」を参照してください。

コピーするソース AMI の AWS リージョン (--source-region) と、新しいカスタム AMI を使用するリージョン (--region) を指定します。source-ami-id は、コピーするイメージの AMI で置き換えてください。source-ami-id は前のステップのコマンドによって返された値です。new-ami-name は、移行先のリージョンの新しい AMI を説明する名前で置き換えます。この手順の後のスクリプトは、source-ami-id の名前の先頭に「Copy of」という文字列を追加して新しい AMI 名を生成します。

```
>aws ec2 copy-image \
    --region us-east-2 \
    --source-image-id source-ami-id \
    --source-region us-east-2 \
    --name new-ami-name
```

3. 環境を更新して、コピーした AMI を使用します。コマンドを実行すると環境のステータスが返されます。

   コマンドウィンドウで、次のようなコマンドを実行します。詳細については、「AWS CLI コマンドリファレンス」の「update-environment」を参照してください。

   更新する必要がある環境とアプリケーションの AWS リージョンを指定します。アプリケーション名と環境名を、前のステップの copied-ami-id に関連付ける必要があるもので置き換えます。--option-setttings パラメータについては、*copied-ami-id* を、前のコマンドの出力からメモした AMI ID で置き換えてください。

```
>aws elasticbeanstalk update-environment \
  --application-name my-application \
  --environment-name my-environment \
  --region us-east-2 \
  --option-settings
 "Namespace=aws:autoscaling:launchconfiguration,OptionName=ImageId,Value=copied-
ami-id"
```

> **Note**
>
> Elastic Beanstalk 環境を起動する必要がなくなった場合は、ストレージコストを最小限に抑えるために AMI をクリーンアップすることを検討してください。詳細については、「カスタム AMI をクリーンアップする」を参照してください。

スタンドアロンスクリプト

次のスクリプトでは、前の手動ステップと同じ結果が得られます。[copy_ami_and_update_env.zip] リンクを選択してスクリプトをダウンロードします。

スクリプトソース: copy_ami_and_update_env.sh

```
#!/bin/bash

set -ue

USAGE="This script is used to copy an AMI used by your Elastic Beanstalk environment
 into your account to use in your environment.\n\n"
USAGE+="Usage:\n\n"
USAGE+="./$(basename $0) [OPTIONS]\n"
USAGE+="OPTIONS:\n"
USAGE+="\t--application-name <application-name>\tThe name of your Elastic Beanstalk
 application.\n"
USAGE+="\t--environment-name <environment-name>\tThe name of your Elastic Beanstalk
 environment.\n"
USAGE+="\t--region <region> \t\t\tThe AWS region your Elastic Beanstalk environment is
 deployed to.\n"
USAGE+="\n\n"
USAGE+="Script Usage Example(s):\n"
USAGE+="./$(basename $0) --application-name my-application --environment-name my-
environment --region us-east-1\n"

if [ $# -eq 0 ]; then
  echo -e $USAGE
  exit
fi

while [[ $# -gt 0 ]]; do
  case $1 in
    --application-name)       APPLICATION_NAME="$2"; shift ;;
```

```
    --environment-name)       ENVIRONMENT_NAME="$2"; shift ;;
    --region)                 REGION="$2"; shift ;;
    *)                        echo "Unknown option $1" ; echo -e $USAGE ; exit ;;
  esac
  shift
done

aws_cli_version="$(aws --version)"
if [ $? -ne 0 ]; then
  echo "aws CLI not found. Please install it: https://docs.aws.amazon.com/cli/latest/
userguide/getting-started-install.html. Exiting."
  exit 1
fi
echo "Using aws CLI version: ${aws_cli_version}"

account=$(aws sts get-caller-identity --query "Account" --output text)
echo "Using account ${account}"

environment_ami_id=$(aws elasticbeanstalk describe-configuration-settings \
  --application-name "$APPLICATION_NAME" \
  --environment-name "$ENVIRONMENT_NAME" \
  --region "$REGION" \
  --query "ConfigurationSettings[0].OptionSettings[?OptionName=='ImageId'] | [0].Value"
 \
  --output text)
echo "Image associated with environment ${ENVIRONMENT_NAME} is ${environment_ami_id}"

owned_image=$(aws ec2 describe-images \
  --owners self \
  --image-ids "$environment_ami_id" \
  --region "$REGION" \
  --query "Images[0]" \
  --output text)
if [ "$owned_image" != "None" ]; then
  echo "${environment_ami_id} is already owned by account ${account}. Exiting."
  exit
fi

source_image_name=$(aws ec2 describe-images \
  --image-ids "$environment_ami_id" \
  --region "$REGION" \
  --query "Images[0].Name" \
  --output text)
if [ "$source_image_name" = "None" ]; then
```

```
  echo "Cannot find ${environment_ami_id}. Please contact AWS support if you need
 additional help: https://aws.amazon.com/support."
  exit 1
fi

copied_image_name="Copy of ${source_image_name}"
copied_ami_id=$(aws ec2 describe-images \
  --owners self \
  --filters Name=name,Values="${copied_image_name}" \
  --region "$REGION" \
  --query "Images[0].ImageId" \
  --output text)
if [ "$copied_ami_id" != "None" ]; then
  echo "Detected that ${environment_ami_id} has already been copied by account
 ${account}. Skipping image copy."
else
  echo "Copying ${environment_ami_id} to account ${account} with name
 ${copied_image_name}"
  copied_ami_id=$(aws ec2 copy-image \
    --source-image-id "$environment_ami_id" \
    --source-region "$REGION" \
    --name "$copied_image_name" \
    --region "$REGION" \
    --query "ImageId" \
    --output text)
  echo "New AMI ID is ${copied_ami_id}"

  echo "Waiting for ${copied_ami_id} to become available"
  aws ec2 wait image-available \
    --image-ids "$copied_ami_id" \
    --region "$REGION"
  echo "${copied_ami_id} is now available"
fi

echo "Updating environment ${ENVIRONMENT_NAME} to use ${copied_ami_id}"
environment_status=$(aws elasticbeanstalk update-environment \
  --application-name "$APPLICATION_NAME" \
  --environment-name "$ENVIRONMENT_NAME" \
  --option-settings
 "Namespace=aws:autoscaling:launchconfiguration,OptionName=ImageId,Value=
${copied_ami_id}" \
  --region "$REGION" \
  --query "Status" \
  --output text)
```

```
echo "Environment ${ENVIRONMENT_NAME} is now ${environment_status}"

echo "Waiting for environment ${ENVIRONMENT_NAME} update to complete"
aws elasticbeanstalk wait environment-updated \
  --application-name "$APPLICATION_NAME" \
  --environment-names "$ENVIRONMENT_NAME" \
  --region "$REGION"
echo "Environment ${ENVIRONMENT_NAME} update complete"
```

> **Note**
>
> スクリプトを実行するには、 AWS CLI がインストールされている必要があります。インストール手順の詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS CLIの最新バージョンを使用してインストールまたは更新を行う」を参照してください。をインストールしたら AWS CLI、環境を所有する AWS アカウントを使用するように設定する必要があります。詳細については、「AWS Command Line Interface ユーザーガイド」の「AWS CLIを設定する」を参照してください。アカウントには、AMI の作成と Elastic Beanstalk 環境の更新を行うアクセス許可も必要です。

次のステップでは、スクリプトが実行するプロセスを説明します。

1. 使用中のアカウントを出力します。
2. 環境 (ソース AMI) が使用している AMI を確認します。
3. ソース AMI が既にアカウントによって所有されているかどうかを確認します。所有されている場合は、終了します。
4. 新しい AMI 名で使用できるように、ソース AMI の名前を決定します。これはソース AMI へのアクセスを確認するためにも使用されます。
5. ソース AMI が既にアカウントにコピーされているかどうかを確認します。この確認は、アカウントが所有するコピー AMI の名前で AMI を検索することによって行います。スクリプト実行の合間に AMI 名が変更された場合、イメージは再度コピーされます。
6. ソース AMI がまだコピーされていない場合は、ソース AMI をアカウントにコピーし、新しい AMI が使用可能になるまで待ちます。
7. 新しい AMI を使用するように環境設定を更新します。
8. 環境の更新が完了するまで待ちます。

copy_ami_and_update_env.zip ファイルからスクリプトを抽出し、次の例を実行してスクリプトを実行します。この例のアプリケーション名と環境名は実際の値で置き換えてください。

```
>sh copy_ami_and_update_env.sh \
  --application-name my-application \
  --environment-name my-environment \
  --region us-east-1
```

> **Note**
>
> Elastic Beanstalk 環境を起動する必要がなくなった場合は、ストレージコストを最小限に抑えるために AMI をクリーンアップすることを検討してください。詳細については、「カスタム AMI をクリーンアップする」を参照してください。

静的ファイルの提供

パフォーマンスを向上させるには、ウェブアプリケーション内のディレクトリセットの静的ファイル (HTML、画像など) を処理するようにプロキシサーバーを設定できます。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接　　処理します。

Elastic Beanstalk では、Amazon Linux 2 に基づいてほとんどのプラットフォームブランチで静的ファイルを配信するようにプロキシを設定できます。1 つの例外は Docker です。

> **Note**
>
> Python および Ruby プラットフォームでは、Elastic Beanstalk はデフォルトでいくつかの静的ファイルフォルダを設定します。詳細については、Python と Ruby の静的ファイル設定セクションを参照してください。このページの説明に従って、追加のフォルダを設定できます。

コンソールを使用した静的ファイルの設定

静的ファイルを処理するようにプロキシサーバーを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. プラットフォームソフトウェア セクションまでスクロールし、静的ファイル グループを見つけます。

   a. 静的ファイルマッピングを追加するには、静的ファイルを追加を選択します。表示される余分な行に、静的ファイルを提供するパスと、提供する静的ファイルを含むディレクトリを入力します。

      - パスフィールで、パス名の先頭にスラッシュ (/) を付けます (/images など)。
      - ディレクトリ フィールドに、アプリケーションのソースコードのルートにあるディレクトリ名を指定します。先頭にスラッシュ (「静的/画像ファイル」など) を付けないでください。

      > **Note**
      > [静的ファイル] セクションが表示されない場合は、設定ファイルを使用して少なくとも 1 つのマッピングを追加する必要があります。詳細については、このページの「the section called “構成オプションを使用した静的ファイルの構成”」を参照してください。

   b. マッピングを削除するには、[削除] を選択します。
6. ページの最下部で [適用] を選択し変更を保存します。

構成オプションを使用した静的ファイルの構成

構成ファイルを使用し、構成オプションを使用して静的ファイルのパスとディレクトリの場所を設定できます。アプリケーションのソースバンドルに設定ファイルを追加し、環境の作成時または後でデプロイするときにデプロイできます。

環境が Amazon Linux 2 に基づくプラットフォームブランチを使用している場合は、`aws:elasticbeanstalk:environment:proxy:staticfiles` 名前空間を使用します。

次の設定ファイルの例は、`/html` パスにある `statichtml` フォルダ内のファイルと、`/images` パスにある `staticimages` フォルダ内のファイルを配信するようにプロキシサーバーに指示します。

Example .ebextensions/static-files.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /html: statichtml
    /images: staticimages
```

Elastic Beanstalk 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、以下の追加情報をお読みください。

Amazon Linux AMI プラットフォーム固有の名前空間

Amazon Linux AMI プラットフォームブランチでは、静的ファイル設定の名前空間はプラットフォームによって異なります。詳細については、次のいずれかのページを参照してください。

- Go 設定の名前空間
- Java SE 設定の名前空間
- Tomcat 設定の名前空間
- Node.js 設定の名前空間
- Python 設定の名前空間

Elastic Beanstalk 環境の HTTPS の設定

このセクションのこのトピックでは、Elastic Beanstalk 環境に HTTPS を設定する方法について説明します。HTTPS は、ユーザーデータやログイン情報を送信するいずれのアプリケーションにも必須です。

Elastic Beanstalk 環境用にカスタムドメイン名を購入して設定した場合は、HTTPS を使用することで、ユーザーがウェブサイトに接続する際の安全性を確保できます。

ドメイン名を所有していない場合でも、開発およびテスト目的に自己署名証明書で、HTTPS を使用できます。詳細については、「サーバー証明書」を参照してください。

ロードバランサーでの HTTPS 終端の設定

ロードバランサーは、アプリケーションを実行する EC2 インスタンスにリクエストを分散します。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。Elastic Beanstalk マルチインスタンス環境で HTTPS を使用する最も簡単な方法は、ロードバラ

ンサーにセキュアなリスナーを設定することです。クライアントとロードバランサー間の接続は安全であるため、HTTPS を終了するようにロードバランサーを設定できます。ロードバランサーと EC2 インスタンスとの間のバックエンド接続では HTTP が使用されるため、インスタンスの追加の設定は必要ありません。セキュアなリスナーを設定する詳細な手順については、「ロードバランサーでの HTTPS 終端の設定」を参照してください。

EC2 インスタンスでの HTTPS 終端の設定

単一インスタンスの環境でアプリケーションを実行したり、ロードバランサーの背後で EC2 インスタンスまで接続をセキュリティで保護する必要がある場合は、HTTPS を終了するように、インスタンス上で実行されるプロキシサーバーを設定できます。HTTPS 接続を終了するようにインスタンスを設定するには、設定ファイルを使用して、インスタンスで実行されるソフトウェアを変更し、セキュアな接続を許可するようにセキュリティグループを変更する必要があります。詳細については、「インスタンスでの HTTPS 終端の設定」を参照してください。

HTTPS エンドツーエンドの設定

負荷分散された環境でのエンドツーエンドの HTTPS の場合、インスタンスとロードバランサーの終了を組み合わせて、両方の接続を暗号化できます。デフォルトでは、HTTPS を使用するトラフィックを転送するようにロードバランサーを設定した場合、ロードバランサーはバックエンドインスタンスによって提示された証明書をすべて信頼します。セキュリティを最大限に高めるには、インスタンスによって提示された公開証明書をロードバランサーが信頼しない場合にそのインスタンスへの接続を禁止するポリシーを、ロードバランサーにアタッチできます。詳細については、「負荷分散された Elastic Beanstalk 環境でエンドツーエンドの暗号化を設定する」を参照してください。

TCP パススルーを使用した HTTPS の設定

また、HTTPS トラフィックを復号せずに中継するように、ロードバランサーを設定することもできます。詳細については、「環境のロードバランサーを TCP パススルー用に設定する」を参照してください。

> **Note**
>
> GitHub のDoes it have Snakes? サンプルアプリケーションには、Tomcat ウェブアプリケーションで HTTPS を設定する方法別に設定ファイルと手順が含まれています。詳細については、readme ファイルと HTTPS に関する手順を参照してください。

トピック

- サーバー証明書
- ロードバランサーでの HTTPS 終端の設定
- インスタンスでの HTTPS 終端の設定
- 負荷分散された Elastic Beanstalk 環境でエンドツーエンドの暗号化を設定する
- 環境のロードバランサーを TCP パススルー用に設定する
- HTTP から HTTPS へのリダイレクトの設定

サーバー証明書

このトピックでは、HTTPS の設定に使用できるさまざまなタイプの証明書と、それぞれを適用するタイミングについて説明します。このセクションのサブトピックでは、独自の証明書を作成する手順と、それをアップロードする方法を説明します。

AWS Certificate Manager (ACM)

ACM は、サーバー証明書をプロビジョン、管理、デプロイするための推奨ツールです。これは、プログラムまたは を使用して行うことができます AWS CLI。ACM を使用すると、ドメイン名用の信頼された証明書を無料で作成できます。

ACM 証明書は AWS ロードバランサーと Amazon CloudFront ディストリビューションでのみ使用でき、ACM は特定の AWS リージョンでのみ使用できます。Elastic Beanstalk で ACM 証明書を使用するには、「ロードバランサーでの HTTPS 終端の設定」を参照してください。ACM の詳細については、「AWS Certificate Manager ユーザーガイド」を参照してください。

> **Note**
>
> ACM を使用できるリージョンのリストについては、「Amazon Web Services 全般のリファレンス」の「ACM のエンドポイントとクォータ」を参照してください。

ACM が AWS リージョンで利用できない場合は、サードパーティーまたは自己署名証明書とプライベートキーを AWS Identity and Access Management (IAM) にアップロードできます。を使用して証明書 AWS CLI をアップロードできます。IAM に保存されている証明書は、ロードバランサーと CloudFront ディストリビューションで使用できます。詳細については、「IAM に証明書をアップロードする」を参照してください。

サードパーティー証明書

ACM がお客様のリージョンで使用できない場合は、信頼された証明書を第三者から購入できます。第三者からの証明書はロードバランサー、バックエンドインスタンス、またはその両方で HTTPS トラフィックの復号に使用できます。

証明書が自己署名です

開発およびテスト目的で、オープンソースのツールを使用して自分で証明書を作成して署名できます。自己署名証明書は無料で簡単に作成できますが、公開サイトのフロントエンド復号化に使用することはできません。クライアントへの HTTPS 接続に自己署名証明書を使用しようとした場合、ユーザーのブラウザではウェブサイトが安全でないことを示すエラーメッセージが表示されます。ただし、バックエンド接続の保護に自己署名証明書を使用することは問題ありません。

X509 証明書を作成し署名する

OpenSSL を使用してアプリケーションの X509 証明書を作成できます。OpenSSL は標準のオープンソースライブラリーで、x509 証明書の作成と署名を含む包括的な暗号関数をサポートしています。OpenSSL の詳細については、www.openssl.org を参照してください。

> **Note**
>
> HTTPS を単一インスタンス環境で使用するか、自己署名証明書を使用して、バックエンドで再度暗号化するには、証明書をローカルに作成する必要があります。ドメイン名を所有している場合は、(AWS Certificate Manager ACM) を使用して で証明書を作成し、ロードバランシングされた環境で無料で AWS 使用できます。手順については、AWS Certificate Manager ユーザーガイドの「証明書のリクエスト」を参照してください。

コマンドラインで `openssl version` を実行して、OpenSSL がすでにインストールされているかどうかを確認します。OpenSSL をインストールしていない場合は、「公開 GitHub リポジトリ」の指示を使用してソースコードをインストールするか、または好みのパッケージマネージャを使用できます。OpenSSL は Elastic Beanstalk の Linux イメージにもインストールされているため、簡単な代替策として、次のように EB CLI の eb ssh コマンドを使用して実行環境の EC2 インスタンスに接続できます。

```
~/eb$ eb ssh
[ec2-user@ip-255-55-55-255 ~]$ openssl version
OpenSSL 1.0.1k-fips 8 Jan 2015
```

証明書署名リクエスト（CSR）を作成するための RSA プライベートキーを作成する必要があります。プライベートキーを作成するには、openssl genrsa コマンドを使用します。

```
[ec2-user@ip-255-55-55-255 ~]$ openssl genrsa 2048 > privatekey.pem
Generating RSA private key, 2048 bit long modulus
.....................................................................................
+++
...............+++
e is 65537 (0x10001)
```

privatekey.pem

プライベートキーを保存するファイルの名前です。通常、openssl genrsa コマンドは画面にプライベートキーの内容を表示しますが、このコマンドは出力をファイルにパイプします。任意のファイル名を選択し、ファイルを後から取得できるように安全な場所に保存します。プライベートキーをなくした場合は、証明書を使用することはできません。

CSR は、デジタルサーバー証明書を申請するために認証機関（CA）に送信するファイルです。CSR を作成するには、openssl req コマンドを使用します。

```
$ openssl req -new -key privatekey.pem -out csr.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

要求された情報を入力して、Enter を押します。以下の表は、各フィールドの例を表示して説明したものです。

名前	説明	例
国名	2 文字の ISO 略称 (国名コード)。	例 :US = アメリカ
州または県	あなたが所属する組織の所在地の州または県。この名前を省略することはできません。	ワシントン
市区町村	あなたが所属する組織の所在地の市区町村。	Seattle

名前	説明	例
組織名	組織の正式名称。組織名は、省略不可です。	Example Corp
部門名	追加の部門情報は、省略可能です。	マーケティング
共通名	ウェブサイトの完全修飾ドメイン名 これは、ユーザーがサイトを訪問したときに表示されるドメイン名と一致する必要があります。一致しない場合は証明書エラーが表示されます。	www.example.com
E メールアドレス	サイト管理者の E メールアドレス	someone@example.com

署名を要求する署名リクエストをサードパーティに送信するか、または開発とテスト用に自分で署名することができます。自己署名証明書は、ロードバランサーと EC2 インスタンス間のバックエンド HTTPS にも使用できます。

証明書に署名するには、openssl x509 コマンドを使用します。以下の例では、以前のステップ (*privatekey.pem*) のプライベートキーおよび署名リクエスト (*csr.pem*) を使用して、*365* 日間有効な *public.crt* という名前の公開証明書を作成します。

```
$ openssl x509 -req -days 365 -in csr.pem -signkey privatekey.pem -out public.crt
Signature ok
subject=/C=us/ST=washington/L=seattle/O=example corporation/OU=marketing/
CN=www.example.com/emailAddress=someone@example.com
Getting Private key
```

秘密キーと公開証明書は後で使用できるようにしておきます。署名リクエストは破棄できます。必ず、プライベートキーは安全な場所に保存し、ソースコードには追加しないことをお勧めします。

Windows Server プラットフォームを使用して証明書を使用するには、PFX 形式に変換する必要があります。プライベートキーとパブリック証明書ファイルから PFX 証明書を作成するには、次のコマンドを使用します。

```
$ openssl pkcs12 -export -out example.com.pfx -inkey privatekey.pem -in public.crt
Enter Export Password: password
Verifying - Enter Export Password: password
```

証明書があるため、ロードバランサーで使用するために、IAM にアップロードするか、HTTPS を終了するように環境内のインスタンスを設定します。

IAM に証明書をアップロードする

Elastic Beanstalk 環境のロードバランサーで証明書を使用するには、証明書とプライベートキーを AWS Identity and Access Management (IAM) にアップロードします。IAM に保存されている証明書は、Elastic Load Balancing ロードバランサーおよび Amazon CloudFront ディストリビューションで使用できます。

> **Note**
>
> AWS Certificate Manager (ACM) は、サーバー証明書のプロビジョニング、管理、デプロイに推奨されるツールです。ACM 証明書のリクエストの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のリクエスト」を参照してください。ACM へのサードパーティー証明書のインポートの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のインポート」を参照してください。IAM を使用して証明書をアップロードするには、AWS リージョンで ACM が使用できない場合に限ります。

AWS Command Line Interface （ AWS CLI) を使用して証明書をアップロードできます。次のコマンドは、*`https-cert.crt`* という名前の自己署名証明書を、*`private-key.pem`* という名前のプライベートキーとともにアップロードします。

```
$ aws iam upload-server-certificate --server-certificate-name elastic-beanstalk-x509 --
certificate-body file://https-cert.crt --private-key file://private-key.pem
{
    "ServerCertificateMetadata": {
        "ServerCertificateId": "AS5YBEIONO2Q7CAIHKNGC",
        "ServerCertificateName": "elastic-beanstalk-x509",
        "Expiration": "2017-01-31T23:06:22Z",
        "Path": "/",
        "Arn": "arn:aws:iam::123456789012:server-certificate/elastic-beanstalk-x509",
        "UploadDate": "2016-02-01T23:10:34.167Z"
    }
}
```

`file://` プレフィックスは、現在のディレクトリにファイルの内容を読み込む AWS CLI ように に指示します。 *`elastic-beanstalk-x509`* は、IAM で証明書を呼び出す名前を指定します。

証明機関から証明書を購入し、証明書チェーンファイルを受け取った場合は、`--certificate-chain` オプションを含めることで、そのファイルもアップロードします。

```
$ aws iam upload-server-certificate --server-certificate-name elastic-beanstalk-x509 --certificate-chain file://certificate-chain.pem --certificate-body file://https-cert.crt --private-key file://private-key.pem
```

証明書の Amazon リソースネーム (ARN) をメモします。これは、HTTPS を使用するロードバランサー設定を更新する際に使用します。

> **Note**
>
> IAM にアップロードされた証明書は、環境のロードバランサーで使用されなくなっても保存されたままになります。これには重要なデータが含まれています。どの環境にも証明書が不要になった場合は、必ず削除してください。IAM から証明書を削除する方法の詳細については、「https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_server-certs.html#delete-server-certificate」を参照してください。

IAM のサーバー証明書の詳細については、「IAM ユーザーガイド」のサーバー証明書の使用に関するページを参照してください。

秘密キーを Amazon S3 に安全に保存する

パブリック証明書の署名に使用するプライベートキーはプライベートであるため、ソースコードにコミットしないでください。プライベートキーファイルを Amazon S3 にアップロードしておき、アプリケーションのデプロイ時に Amazon S3 からダウンロードされるように Elastic Beanstalk を設定することで、設定ファイルにプライベートキーを保存しなくて済みます。

以下の例で示している設定ファイルのリソースセクションとファイルセクションでは、Amazon S3 バケットからプライベートキーファイルをダウンロードしています。

Example .ebextensions/privatekey.config

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["elasticbeanstalk-us-west-2-123456789012"]
```

```
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"
files:
  # Private key
  "/etc/pki/tls/certs/server.key":
    mode: "000400"
    owner: root
    group: root
    authentication: "S3Auth"
    source: https://elasticbeanstalk-us-west-2-123456789012.s3.us-west-2.amazonaws.com/
server.key
```

例に使用しているバケット名と URL は独自のものに置き換えてください。このファイルの最初のエントリは、環境の Auto Scaling グループのメタデータに `S3Auth` という名前の認証方法を追加します。お客様の環境用にカスタムインスタンスプロファイルを設定している場合はそのプロファイルが使用されますが、設定していない場合は `aws-elasticbeanstalk-ec2-role` のデフォルト値が適用されます。デフォルトのインスタンスプロファイルには、Elastic Beanstalk ストレージバケットからの読み取り権限があります。別のバケットを使用する場合は、インスタンスプロファイルに許可を追加します。

2 番目のエントリは、`S3Auth` 認証方法を使用して、指定された URL からプライベートキーをダウンロードし、`/etc/pki/tls/certs/server.key` に保存します。プロキシサーバーは、この場所からプライベートキーを読み取って、インスタンスで HTTPS 接続を終了できます。

環境の EC2 インスタンスに割り当てられるインスタンスプロファイルには、指定したバケットからキーオブジェクトを読み取るための権限がなければなりません。インスタンスプロファイルに IAM のオブジェクトを読み取るアクセス許可があることと、バケットおよびオブジェクトに対するアクセス許可でインスタンスプロファイルが禁止されていないことを確認します。

バケットの権限を表示するには

1. Amazon S3 マネジメントコンソールを開きます。
2. バケットを選択します。
3. [プロパティ] を選択して、[アクセス許可] を選択します。
4. アカウントがバケットの読み取り権限を持っていることを確認します。
5. バケットポリシーがアタッチされている場合は、[バケットポリシー] を選択して、バケットに割り当てられているアクセス許可を表示します。

ロードバランサーでの HTTPS 終端の設定

HTTPS を使用するように AWS Elastic Beanstalk 環境を更新するには、環境でロードバランサーの HTTPS リスナーを設定する必要があります。HTTPS リスナーは、Classic Load Balancer と Application Load Balancer の 2 種類のロードバランサーをサポートします。

Elastic Beanstalk コンソールまたは設定ファイルのいずれかを使用して、セキュアリスナーを設定し、証明書を割り当てることができます。

> **Note**
>
> 単一インスタンス環境にはロードバランサーが存在せず、ロードバランサーでは HTTPS 接尾辞はサポートされません。

Elastic Beanstalk コンソールを使用したセキュアリスナーの設定

お客様の環境のロードバランサーに証明書を割り当てるには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [ロードバランサー] 設定カテゴリで、[編集] を選択します。

   > **Note**
   >
   > [ロードバランサー] 設定カテゴリに [編集] ボタンがない場合、お客様の環境にはロードバランサーがありません。

5. [ロードバランサーの変更] ページの手順は、環境に関連付けられるロードバランサーのタイプによって異なります。
   - Classic Load Balancer
     a. [リスナーの追加] を選択します。
     b. [Classic Load Balancer リスナー] ダイアログボックスで次の設定を構成します。
        - [リスナーポート] に着信トラフィックポートを入力します (通常の場合は、443)。
        - [リスナープロトコル] で [HTTPS] を選択します。

- [インスタンスポート] に 80 と入力します。
- [インスタンスのプロトコル] で [HTTP] を選択します。
- [SSL 証明書] で、証明書を選択します。

c. [Add] (追加) を選択します。

- Application Load Balancer

a. [リスナーの追加] を選択します。

b. [Application Load Balancer リスナー] ダイアログボックスで次の設定を構成します。

- [ポート] で着信トラフィックポートを入力します (通常の場合は、443)。
- [プロトコル] として [HTTPS] を選択します。
- [SSL 証明書] で、証明書を選択します。

c. [Add] (追加) を選択します。

> **Note**
>
> Classic Load Balancer および Application Load Balancer について、ドロップダウン・メニューに証明書が表示されない場合は、AWS Certificate Manager (ACM) でカスタムドメイン名の証明書を作成またはアップロードします (推奨)。または、AWS CLIで 証明書を IAM にアップロードします。

- Network Load Balancer

a. [リスナーの追加] を選択します。

b. [Network Load Balancer リスナー] ダイアログボックスで、[ポート] に着信トラフィックポート (通常は 443) を入力します。

c. [Add] (追加) を選択します。

6. ページの最下部で [適用] を選択し変更を保存します。

設定ファイルを使用したセキュアリスナーの設定

次のいずれかの 設定ファイルを使用して、ロードバランサーに対してセキュアなリスナーを設定できます。

Example .ebextensions/securelistener-clb.config

この例は、環境に Classic Load Balancer がある場合に使用します。この例では、 指定した証明書を使用して ポート 443 の HTTPS リスナーを設定し、復号化されたトラフィックがポート 80 上でお客様の環境内のインスタンスに転送されるように、`aws:elb:listener` 名前空間のオプションを使用します。

```
option_settings:
  aws:elb:listener:443:
    SSLCertificateId: arn:aws:acm:us-east-2:1234567890123:certificate/
####################################
    ListenerProtocol: HTTPS
    InstancePort: 80
```

強調表示されたテキストを証明書の ARN に置き換えます。証明書は、 AWS Certificate Manager (ACM) で作成またはアップロードした証明書 (推奨) でも、 を使用して IAM にアップロードした証明書でもかまいません AWS CLI。

Classic Load Balancer の設定オプションの詳細については、「Classic Load Balancer 設定の名前空間」を参照してください。

Example .ebextensions/securelistener-alb.config

この例は、環境に Application Load Balancer がある場合に使用します。この例では、`aws:elbv2:listener` 名前空間内のオプションを使用して、指定した証明書によりポート 443 上の HTTPS リスナーを設定しています。リスナーは、デフォルトのプロセスにトラフィックをルーティングします。

```
option_settings:
  aws:elbv2:listener:443:
    ListenerEnabled: 'true'
    Protocol: HTTPS
    SSLCertificateArns: arn:aws:acm:us-east-2:1234567890123:certificate/
####################################
```

Example .ebextensions/securelistener-nlb.config

この例は、環境に Network Load Balancer がある場合に使用します。この例では、`aws:elbv2:listener` 名前空間内のオプションを使用して、ポート 443 のリスナーを設定します。リスナーは、デフォルトのプロセスにトラフィックをルーティングします。

```
option_settings:
  aws:elbv2:listener:443:
    ListenerEnabled: 'true'
```

セキュリティグループを設定する

ポート 80 以外のインスタンスポートにトラフィックを転送するようにロードバランサーを設定する場合は、ロードバランサーからのインスタンスポート上のインバウンドトラフィックを許可するルールをセキュリティグループに追加する必要があります。カスタム VPC で環境を作成すると、Elastic Beanstalk によってこのルールが自動的に追加されます。

このルールは、`Resources` キーを、アプリケーションの `.ebextensions` ディレクトリにある設定ファイルに追加することで追加できます。

次の設定ファイルの例では、`AWSEBSecurityGroup` セキュリティグループに進入ルールを追加します。これにより、ロードバランサーのセキュリティグループからポート 1000 へのトラフィックが許可されます。

Example .ebextensions/sg-ingressfromlb.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 1000
      FromPort: 1000
      SourceSecurityGroupId: {"Fn::GetAtt" : ["AWSEBLoadBalancerSecurityGroup",
 "GroupId"]}
```

インスタンスでの HTTPS 終端の設定

設定ファイルを使用して、アプリケーションにトラフィックを渡すプロキシサーバーを設定し、HTTPS 接続を終了できます。これは、単一インスタンス環境で HTTPS を使用している場合、またはトラフィックを復号化しないで渡すようにロードバランサーを設定している場合に便利です。

HTTPS を有効にするには、Elastic Beanstalk アプリケーションが実行されている EC2 インスタンスにポート 443 で受信トラフィックを許可する必要があります。このために、設定ファイルの `Resources` キーを使用して、AWSEBSecurityGroup の受信ルールにポート 443 のルールを追加します。

次の例は、AWSEBSecurityGroup セキュリティグループに受信ルールを追加して、単一インスタンス環境のセキュリティグループのすべてのトラフィック用にポート 443 を開きます。

.ebextensions/https-instance-securitygroup.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

デフォルト Amazon Virtual Private Cloud (Amazon VPC) のロードバランシング環境では、ロードバランサーからのトラフィックのみを受け入れるように、このポリシーを変更できます。例については、「負荷分散された Elastic Beanstalk 環境でエンドツーエンドの暗号化を設定する」を参照してください。

[Platforms] (プラットフォーム)

- Docker を実行している EC2 インスタンスでの HTTPS の終了
- Go を実行している EC2 インスタンスでの HTTPS の終了
- Java SE を実行する EC2 インスタンスで HTTPS を終了する
- Node.js を実行している EC2 インスタンスで HTTPS を終了する
- PHP を実行している EC2 インスタンスでの HTTPS の終了
- Python を実行している EC2 インスタンスの HTTPS を終了する
- Ruby を実行している EC2 インスタンスでの HTTPS の終了
- Tomcat を実行している EC2 インスタンスでの HTTPS の終了
- .NET Core on Linux を実行している Amazon EC2 インスタンスでの HTTPS の終了
- .NET を実行している Amazon EC2 インスタンスでの HTTPS の終了

Docker を実行している EC2 インスタンスでの HTTPS の終了

Docker コンテナでは、設定ファイルを使用して HTTPS を有効にします。

次のスニペットを設定ファイルに追加して、証明書とプライベートキー資料を説明に沿って置き換え、ソースバンドルの `.ebextensions` ディレクトリに保存します。設定ファイルは以下のタスクを実行します。

- `files` キーはインスタンスに次のファイルを作成します。

  `/etc/nginx/conf.d/https.conf`

  nginx サーバーを設定します。このファイルは、nginx サービスの開始時にロードされます。

  `/etc/pki/tls/certs/server.crt`

  インスタンスに証明書ファイルを作成します。*##########*をお客様の証明書の内容に置き換えます。

  > **Note**
  > YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

  中間証明書がある場合は、`server.crt` のサイト証明書の後に組み込みます。

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

  `/etc/pki/tls/certs/server.key`

  インスタンスにプライベートキーのファイルを作成します。*###########*を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

Example .ebextensions/https-instance.config

```
files:
  /etc/nginx/conf.d/https.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      # HTTPS Server

      server {
        listen 443;
        server_name localhost;

        ssl on;
        ssl_certificate /etc/pki/tls/certs/server.crt;
        ssl_certificate_key /etc/pki/tls/certs/server.key;

        ssl_session_timeout 5m;

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;

        location / {
          proxy_pass http://docker;
          proxy_http_version 1.1;

          proxy_set_header Connection "";
          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_set_header X-Forwarded-Proto https;
        }
      }

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----
```

```
  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----
```

> **Note**
>
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

Go を実行している EC2 インスタンスでの HTTPS の終了

Go コンテナタイプでは、設定ファイルと nginx 設定ファイルで HTTPS を有効にして、nginx サーバーが HTTPS を使用するように設定します。

次のスニペットを設定ファイルに追加して、証明書とプライベートキープレースホルダーを説明に沿って置き換え、ソースバンドルの `.ebextensions` ディレクトリに保存します。設定ファイルは以下のタスクを実行します。

- `Resources` キーは、環境のインスタンスによって使用されるセキュリティグループでポート 443 を有効にします。
- `files` キーはインスタンスに次のファイルを作成します。

  `/etc/pki/tls/certs/server.crt`

  インスタンスに証明書ファイルを作成します。*##########*をお客様の証明書の内容に置き換えます。

  > **Note**
  >
  > YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

  中間証明書がある場合は、`server.crt` のサイト証明書の後に組み込みます。

  ```
      -----BEGIN CERTIFICATE-----
  certificate file contents
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  first intermediate certificate
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  second intermediate certificate
  -----END CERTIFICATE-----
  ```

  `/etc/pki/tls/certs/server.key`

  インスタンスにプライベートキーのファイルを作成します。*###########*を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

- `container_commands` キーは、すべての設定が完了してから nginx サーバーを再起動することで、サーバーが nginx 設定ファイルを読み込みます。

Example .ebextensions/https-instance.config

```
files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "service nginx restart"
```

> **Note**
>
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

ソースバンドルの `.conf` ディレクトリの `.ebextensions/nginx/conf.d/` 拡張子が付いたファイルに以下を格納します (たとえば、`.ebextensions/nginx/conf.d/https.conf`)。*app_port* を、アプリケーションがリッスンするポート番号に置き換えます。この例は、SSL を使用してポート番号 443 を使用するように nginx サーバーを設定しています。Go プラットフォームの設定ファイルについての詳細は、「プロキシサーバーを設定します」を参照してください。

Example .ebextensions/nginx/conf.d/https.conf

```
# HTTPS server

server {
    listen       443;
    server_name  localhost;
```

```
    ssl                  on;
    ssl_certificate      /etc/pki/tls/certs/server.crt;
    ssl_certificate_key  /etc/pki/tls/certs/server.key;

    ssl_session_timeout  5m;

    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers   on;

    location / {
        proxy_pass  http://localhost:app_port;
        proxy_set_header   Connection "";
        proxy_http_version 1.1;
        proxy_set_header        Host            $host;
        proxy_set_header        X-Real-IP       $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Forwarded-Proto https;
    }
}
```

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

Java SE を実行する EC2 インスタンスで HTTPS を終了する

Java SE コンテナタイプでは、.ebextensions 設定ファイルで HTTPS を有効にして、nginx 設定ファイルで HTTPS を使用するように nginx サーバーを設定します。

すべての AL2023/AL2 プラットフォームでは、統一されたプロキシ設定機能がサポートされています。AL2023/AL2 を実行中のプラットフォームバージョンでプロキシサーバーを設定する方法の詳細については、「リバースプロキシの設定」を参照してください。

次のスニペットを設定ファイルに追加して、証明書とプライベートキープレースホルダーを説明に沿って置き換え、`.ebextensions` ディレクトリに保存します。設定ファイルは以下のタスクを実行します。

- `files` キーはインスタンスに次のファイルを作成します。

  `/etc/pki/tls/certs/server.crt`

  インスタンスに証明書ファイルを作成します。`##########`をお客様の証明書の内容に置き換えます。

  > **Note**
  >
  > YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

  中間証明書がある場合は、`server.crt` のサイト証明書の後に組み込みます。

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

インスタンスにプライベートキーのファイルを作成します。*##########*を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

- `container_commands` キーは、すべての設定が完了してから nginx サーバーを再起動することで、サーバーが nginx 設定ファイルを読み込みます。

Example .ebextensions/https-instance.config

```
files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "service nginx restart"
```

> **Note**
>
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

ソースバンドルの `.conf` ディレクトリの `.ebextensions/nginx/conf.d/` 拡張子が付いたファイルに以下を格納します (たとえば、`.ebextensions/nginx/conf.d/https.conf`)。*app_port* を、アプリケーションがリッスンするポート番号に置き換えます。この例は、SSL を使用してポート番号 443 を使用するように nginx サーバーを設定しています。Java SE プラットフォームの設定についての詳細は「プロキシサーバーを設定します」を参照してください。

Example .ebextensions/nginx/conf.d/https.conf

```
# HTTPS server

server {
    listen       443;
    server_name  localhost;

    ssl                  on;
    ssl_certificate      /etc/pki/tls/certs/server.crt;
    ssl_certificate_key  /etc/pki/tls/certs/server.key;

    ssl_session_timeout  5m;

    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers   on;

    location / {
        proxy_pass  http://localhost:app_port;
        proxy_set_header   Connection "";
        proxy_http_version 1.1;
        proxy_set_header        Host            $host;
        proxy_set_header        X-Real-IP       $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Forwarded-Proto https;
    }
}
```

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
```

```
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

Node.js を実行している EC2 インスタンスで HTTPS を終了する

次の設定ファイルの例では、デフォルトの nginx 設定を拡張して、ポート 443 をリッスンし、公開証明書とプライベートキーで SSL/TLS 接続を終了します。

拡張ヘルスレポートの環境を設定した場合は、アクセスログを生成するように nginx を設定する必要があります。そのためには、先頭の `# For enhanced health...` 文字を削除して、# というコメントの下にある行のブロックをコメント解除します。

Example .ebextensions/https-instance.config

```
files:
  /etc/nginx/conf.d/https.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      # HTTPS server

      server {
          listen       443;
          server_name  localhost;

          ssl                  on;
          ssl_certificate      /etc/pki/tls/certs/server.crt;
          ssl_certificate_key  /etc/pki/tls/certs/server.key;

          ssl_session_timeout  5m;

          ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
          ssl_prefer_server_ciphers   on;

          # For enhanced health reporting support, uncomment this block:

          #if ($time_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2})") {
          #    set $year $1;
          #    set $month $2;
```

```
        #    set $day $3;
        #    set $hour $4;
        #}
        #access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour
healthd;
        #access_log  /var/log/nginx/access.log  main;

        location / {
            proxy_pass  http://nodejs;
            proxy_set_header   Connection "";
            proxy_http_version 1.1;
            proxy_set_header        Host            $host;
            proxy_set_header        X-Real-IP       $remote_addr;
            proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header        X-Forwarded-Proto https;
        }
    }

 /etc/pki/tls/certs/server.crt:
   mode: "000400"
   owner: root
   group: root
   content: |
     -----BEGIN CERTIFICATE-----
     certificate file contents
     -----END CERTIFICATE-----

 /etc/pki/tls/certs/server.key:
   mode: "000400"
   owner: root
   group: root
   content: |
     -----BEGIN RSA PRIVATE KEY-----
     private key contents # See note below.
     -----END RSA PRIVATE KEY-----
```

`files` キーはインスタンスに次のファイルを作成します。

`/etc/nginx/conf.d/https.conf`

nginx サーバーを設定します。このファイルは、nginx サービスの開始時にロードされます。

`/etc/pki/tls/certs/server.crt`

インスタンスに証明書ファイルを作成します。*##########*をお客様の証明書の内容に置き換えます。

> **Note**
>
> YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

中間証明書がある場合は、`server.crt` のサイト証明書の後に組み込みます。

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

インスタンスにプライベートキーのファイルを作成します。*###########*を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

> **Note**
>
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

PHP を実行している EC2 インスタンスでの HTTPS の終了

PHP コンテナタイプの場合、設定ファイルを使用して、Apache HTTP Server が HTTPS を使用できるようにします。

次のスニペットを設定ファイルに追加して、証明書とプライベートキー資料を説明に沿って置き換え、ソースバンドルの `.ebextensions` ディレクトリに保存します。

設定ファイルは以下のタスクを実行します。

- `packages` キーは、yum を使用して `mod24_ssl` をインストールします。
- `files` キーはインスタンスに次のファイルを作成します。

  `/etc/httpd/conf.d/ssl.conf`

  Apache サーバーを設定します。このファイルは、Apache サービスの開始時期をロードします。

  `/etc/pki/tls/certs/server.crt`

  インスタンスに証明書ファイルを作成します。`##########`をお客様の証明書の内容に置き換えます。

> **Note**
>
> YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

中間証明書がある場合は、`server.crt` のサイト証明書の後に組み込みます。

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

インスタンスにプライベートキーのファイルを作成します。`##########`を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

Example .ebextensions/https-instance.config

```
packages:
  yum:
    mod24_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      LoadModule ssl_module modules/mod_ssl.so
      Listen 443
      <VirtualHost *:443>
        <Proxy *>
```

```
          Order deny,allow
          Allow from all
        </Proxy>

        SSLEngine             on
        SSLCertificateFile    "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"
        SSLCipherSuite        EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
        SSLProtocol           All -SSLv2 -SSLv3
        SSLHonorCipherOrder   On
        SSLSessionTickets     Off

        Header always set Strict-Transport-Security "max-age=63072000;
includeSubdomains; preload"
        Header always set X-Frame-Options DENY
        Header always set X-Content-Type-Options nosniff

        ProxyPass / http://localhost:80/ retry=0
        ProxyPassReverse / http://localhost:80/
        ProxyPreserveHost on
        RequestHeader set X-Forwarded-Proto "https" early

      </VirtualHost>

 /etc/pki/tls/certs/server.crt:
   mode: "000400"
   owner: root
   group: root
   content: |
     -----BEGIN CERTIFICATE-----
     certificate file contents
     -----END CERTIFICATE-----

 /etc/pki/tls/certs/server.key:
   mode: "000400"
   owner: root
   group: root
   content: |
     -----BEGIN RSA PRIVATE KEY-----
     private key contents # See note below.
     -----END RSA PRIVATE KEY-----
```

> **Note**
>
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

Python を実行している EC2 インスタンスの HTTPS を終了する

Web Server Gateway Interface (WSGI) で Apache HTTP Server を使用する Python コンテナタイプの場合、設定ファイルを使用して、Apache HTTP Server が HTTPS を使用できるようにします。

次のスニペットを設定ファイルに追加して、証明書とプライベートキーマテリアルを説明に従って置き換え、ソースバンドルの `.ebextensions` ディレクトリに保存します。設定ファイルは以下のタスクを実行します。

- `packages` キーは、yum を使用して `mod24_ssl` をインストールします。
- `files` キーはインスタンスに次のファイルを作成します。

`/etc/httpd/conf.d/ssl.conf`

Apache サーバーを設定します。アプリケーションの名前が `application.py` でない場合は、`WSGIScriptAlias` の値内の強調表示されたテキストをそのアプリケーションへのローカルパスに置き換えてください。たとえば、django アプリケーションのものは `django/wsgi.py` にあることがあります。この場所は、お客様の環境用に設定した `WSGIPath` オプションの値と一致する必要があります。

アプリケーションの要件によっては、python-path パラメータに他のディレクトリを追加する必要もあります。

`/etc/pki/tls/certs/server.crt`

インスタンスに証明書ファイルを作成します。*##########*をお客様の証明書の内容に置き換えます。

> **Note**
>
> YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

中間証明書がある場合は、`server.crt` のサイト証明書の後に組み込みます。

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

インスタンスにプライベートキーのファイルを作成します。*###########*を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

- `container_commands` キーは、すべての設定が完了してから httpd サービスを停止し、httpd サービスが新しい `https.conf` ファイルと証明書を使用できるようにします。

> **Note**
>
> この例は、Python プラットフォームを使用する環境のみで動作します。

Example .ebextensions/https-instance.config

```
packages:
  yum:
    mod24_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      LoadModule wsgi_module modules/mod_wsgi.so
      WSGIPythonHome /opt/python/run/baselinenv
      WSGISocketPrefix run/wsgi
      WSGIRestrictEmbedded On
      Listen 443
      <VirtualHost *:443>
        SSLEngine on
        SSLCertificateFile "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"

        Alias /static/ /opt/python/current/app/static/
        <Directory /opt/python/current/app/static>
        Order allow,deny
        Allow from all
        </Directory>

        WSGIScriptAlias / /opt/python/current/app/application.py

        <Directory /opt/python/current/app>
        Require all granted
        </Directory>
```

```
        WSGIDaemonProcess wsgi-ssl processes=1 threads=15 display-name=%{GROUP} \
          python-path=/opt/python/current/app \
          python-home=/opt/python/run/venv \
          home=/opt/python/current/app \
          user=wsgi \
          group=wsgi
        WSGIProcessGroup wsgi-ssl

      </VirtualHost>

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01killhttpd:
    command: "killall httpd"
  02waitforhttpddeath:
    command: "sleep 3"
```

> **Note**
>
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

Ruby を実行している EC2 インスタンスでの HTTPS の終了

Ruby コンテナタイプの場合、HTTPS を有効にする方法は使用するアプリケーションサーバーの種類によって異なります。

トピック

- Puma を使用する Ruby 用の HTTPS を設定する
- Passenger を使用する Ruby 用の HTTPS を設定する

Puma を使用する Ruby 用の HTTPS を設定する

アプリケーションサーバーとして Puma を使用する Ruby コンテナタイプの場合、設定ファイルを使用して HTTPS を有効にします。

次のスニペットを設定ファイルに追加して、証明書とプライベートキー資料を説明に沿って置き換え、ソースバンドルの `.ebextensions` ディレクトリに保存します。設定ファイルは以下のタスクを実行します。

- `files` キーはインスタンスに次のファイルを作成します。

`/etc/nginx/conf.d/https.conf`

nginx サーバーを設定します。このファイルは、nginx サービスの開始時にロードされます。

`/etc/pki/tls/certs/server.crt`

インスタンスに証明書ファイルを作成します。`##########`をお客様の証明書の内容に置き換えます。

> **Note**
>
> YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

中間証明書がある場合は、`server.crt` のサイト証明書の後に組み込みます。

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

インスタンスにプライベートキーのファイルを作成します。`###########`を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

- `container_commands` キーは、すべての設定が完了してから nginx サーバーを再起動することで、サーバーが新しい `https.conf` ファイルを使用できるようにします。

Example .ebextensions/https-instance.config

```
files:
  /etc/nginx/conf.d/https.conf:
    content: |
```

```
    # HTTPS server

    server {
        listen       443;
        server_name  localhost;

        ssl                  on;
        ssl_certificate      /etc/pki/tls/certs/server.crt;
        ssl_certificate_key  /etc/pki/tls/certs/server.key;

        ssl_session_timeout  5m;

        ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers   on;

        location / {
            proxy_pass  http://my_app;
            proxy_set_header        Host            $host;
            proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header        X-Forwarded-Proto https;
        }

        location /assets {
          alias /var/app/current/public/assets;
          gzip_static on;
          gzip on;
          expires max;
          add_header Cache-Control public;
        }

        location /public {
          alias /var/app/current/public;
          gzip_static on;
          gzip on;
          expires max;
          add_header Cache-Control public;
        }
    }

/etc/pki/tls/certs/server.crt:
  content: |
    -----BEGIN CERTIFICATE-----
    certificate file contents
    -----END CERTIFICATE-----
```

```
  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "service nginx restart"
```

> **Note**
>
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

Passenger を使用する Ruby 用の HTTPS を設定する

アプリケーションサーバーとして Passenger を使用する Ruby コンテナタイプの場合、設定ファイルと JSON ファイルの両方を使用してHTTPS を有効にします。

Passenger を使用する Ruby 用の HTTPS を設定する方法

1. 次のスニペットを設定ファイルに追加して、証明書とプライベートキー資料を説明に沿って置き換え、ソースバンドルの `.ebextensions` ディレクトリに保存します。設定ファイルは以下のタスクを実行します。

   - `files` キーはインスタンスに次のファイルを作成します。

     `/etc/pki/tls/certs/server.crt`

     インスタンスに証明書ファイルを作成します。*##########*をお客様の証明書の内容に置き換えます。

     > **Note**
     >
     > YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

     中間証明書がある場合は、`server.crt` のサイト証明書の後に組み込みます。

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

インスタンスにプライベートキーのファイルを作成します。`##########`を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

Example Passenger を使用する Ruby 用 HTTPS を設定するための .Ebextensions スニペット

```
files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----
```

> **Note**
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

2. テキストファイルを作成して、ファイルに次の JSON を追加します。作成したファイルを、`passenger-standalone.json` という名前でソースバンドルのルートディレクトリに保存します。この JSON ファイルは、Passenger が HTTPS を使用するように設定します。

> **Important**
> この JSON ファイルには、バイト順マーク (BOM) が含まれてはいけません。BOM が含まれていると、Passenger JSON ライブラリはファイルを正しく読み取れず、Passenger サービスは開始されません。

Example passenger-standalone.json

```
{
  "ssl" : true,
  "ssl_port" : 443,
  "ssl_certificate" : "/etc/pki/tls/certs/server.crt",
  "ssl_certificate_key" : "/etc/pki/tls/certs/server.key"
}
```

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

Tomcat を実行している EC2 インスタンスでの HTTPS の終了

Tomcat コンテナタイプでは、設定ファイルを使用して、Tomcat 用のリバースプロキシとして機能する際に Apache HTTP サーバーが HTTPS を使用できるようにします。

次のスニペットを設定ファイルに追加して、証明書とプライベートキー資料を説明に沿って置き換え、ソースバンドルの `.ebextensions` ディレクトリに保存します。設定ファイルは以下のタスクを実行します。

- `files` キーはインスタンスに次のファイルを作成します。

`/etc/pki/tls/certs/server.crt`

インスタンスに証明書ファイルを作成します。##########をお客様の証明書の内容に置き換えます。

> **Note**
>
> YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

`/etc/pki/tls/certs/server.key`

インスタンスにプライベートキーのファイルを作成します。###########を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

`/opt/elasticbeanstalk/hooks/appdeploy/post/99_start_httpd.sh`

httpd サービスを再起動するためのデプロイメント後のフックスクリプトを作成します。

Example .ebextensions/https-instance.config

```
files:
  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----
```

```
  /opt/elasticbeanstalk/hooks/appdeploy/post/99_start_httpd.sh:
    mode: "000755"
    owner: root
    group: root
    content: |
      #!/usr/bin/env bash
      sudo service httpd restart
```

ポート 443 をリッスンするように、環境のプロキシサーバーも設定する必要があります。次の Apache 2.4 設定では、ポート 443 にリスナーを追加します。詳細については、「プロキシサーバーを設定します」を参照してください。

Example .ebextensions/httpd/conf.d/ssl.conf

```
Listen 443
<VirtualHost *:443>
  ServerName server-name
  SSLEngine on
  SSLCertificateFile "/etc/pki/tls/certs/server.crt"
  SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"

  <Proxy *>
    Require all granted
  </Proxy>
  ProxyPass / http://localhost:8080/ retry=0
  ProxyPassReverse / http://localhost:8080/
  ProxyPreserveHost on

  ErrorLog /var/log/httpd/elasticbeanstalk-ssl-error_log

</VirtualHost>
```

証明書ベンダーには、モバイルクライアントとの互換性向上のためにインストールできる中間証明書が含まれる場合があります。SSL 設定ファイルに以下を追加することで、中間認証局 (CA) バンドルを使用して Apache を設定します (場所については「デフォルトの Apache 設定の拡張および上書き — Amazon Linux AMI (AL1)」を参照)。

- `ssl.conf` ファイルコンテンツの場合、次の連鎖ファイルを指定:

  ```
  SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"
  ```

```
SSLCertificateChainFile "/etc/pki/tls/certs/gd_bundle.crt"
SSLCipherSuite        EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
```

- 中間証明書のコンテンツを使用して `files` キーに新しいエントリを追加します。

```
files:
  /etc/pki/tls/certs/gd_bundle.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      First intermediate certificate
      -----END CERTIFICATE-----
      -----BEGIN CERTIFICATE-----
      Second intermediate certificate
      -----END CERTIFICATE-----
```

> **Note**
>
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
```

```
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

.NET Core on Linux を実行している Amazon EC2 インスタンスでの HTTPS の終了

.NET Core on Linux コンテナタイプでは、`.ebextensions`設定ファイルで HTTPS を有効にして、nginx 設定ファイルで HTTPS を使用するように nginx サーバーを設定します。

次のスニペットを設定ファイルに追加して、証明書とプライベートキープレースホルダーを説明に沿って置き換え、`.ebextensions` ディレクトリに保存します。設定ファイルは以下のタスクを実行します。

- `files` キーはインスタンスに次のファイルを作成します。

  `/etc/pki/tls/certs/server.crt`

  インスタンスに証明書ファイルを作成します。`##########`をお客様の証明書の内容に置き換えます。

  > **Note**
  >
  > YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

  中間証明書がある場合は、`server.crt` のサイト証明書の後に組み込みます。

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

インスタンスにプライベートキーのファイルを作成します。`##########`を、証明書リクエストまたは自己署名証明書の作成に使用したプライベートキーの内容に置き換えます。

- `container_commands` キーは、すべての設定が完了してから nginx サーバーを再起動することで、サーバーが nginx 設定ファイルを読み込みます。

Example .ebextensions/https-instance.config

```
files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "systemctl restart nginx"
```

> **Note**
>
> プライベートキーを含む設定ファイルがソースコントロールにコミットされないようにしてください。設定をテストして動作が適切であることを確認したら、プライベートキーを Amazon S3 に保存して、デプロイ中にダウンロードされるように設定を変更します。手順については、秘密キーを Amazon S3 に安全に保存する を参照してください。

ソースバンドルの `.conf` ディレクトリの `.platform/nginx/conf.d/` 拡張子が付いたファイルに以下を格納します (たとえば、`.platform/nginx/conf.d/https.conf`)。*`app_port`* を、アプリケーションがリッスンするポート番号に置き換えます。この例は、SSL を使用してポート番号 443 を使用するように nginx サーバーを設定しています。.NET Core on Linux プラットフォームの設定ファイルについての詳細は、「the section called “プロキシサーバー”」を参照してください。

Example .platform/nginx/conf.d/https.conf

```
# HTTPS server

server {
    listen       443 ssl;
    server_name  localhost;

    ssl_certificate      /etc/pki/tls/certs/server.crt;
    ssl_certificate_key  /etc/pki/tls/certs/server.key;

    ssl_session_timeout  5m;

    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers   on;

    location / {
        proxy_pass  http://localhost:app_port;
        proxy_set_header   Connection "";
        proxy_http_version 1.1;
        proxy_set_header        Host            $host;
        proxy_set_header        X-Real-IP       $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Forwarded-Proto https;
    }
}
```

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

.NET を実行している Amazon EC2 インスタンスでの HTTPS の終了

次の設定ファイルは、以下のタスクを実行する Windows PowerShell スクリプトを作成して実行します。

- ポート 443 にバインドされる既存の HTTPS 証明書を確認する。
- Amazon S3 バケットから、PFX 証明書とパスワードを取得する。

> **Note**
> Amazon S3 バケット内の SSL 証明書にアクセスするには、`AmazonS3ReadOnlyAccess` ポリシーを `aws-elasticbeanstalk-ec2-role` に追加します。

- パスワードの取得元 AWS Secrets Manager。

> **Note**
> 証明書パスワードを含むシークレットの `secretsmanager:GetSecretValue` アクションを許可するステートメントを `aws-elasticbeanstalk-ec2-role` に追加します

- 証明書をインストールする。
- ポート 443 に証明書をバインドする。

> **Note**
> HTTP エンドポイント (ポート 80) を削除するには、`Remove-WebBinding` コマンドをサンプルの「HTTP バインディングを削除する」セクションに含めます。

Example .ebextensions/https-instance-dotnet.config

```
files:
  "C:\\certs\\install-cert.ps1":
    content: |
      import-module webadministration
      ## Settings - replace the following values with your own
```

```
      $bucket = "amzn-s3-demo-bucket"  ## S3 bucket name
      $certkey = "example.com.pfx"    ## S3 object key for your PFX certificate
      $secretname = "example_secret"  ## AWS Secrets Manager name for a secret that
 contains the certificate's password
      ##

      # Set variables
      $certfile = "C:\cert.pfx"
      $pwd = Get-SECSecretValue -SecretId $secretname | select -expand SecretString

      # Clean up existing binding
      if ( Get-WebBinding "Default Web Site" -Port 443 ) {
        Echo "Removing WebBinding"
        Remove-WebBinding -Name "Default Web Site" -BindingInformation *:443:
      }
      if ( Get-Item -path IIS:\SslBindings\0.0.0.0!443 ) {
        Echo "Deregistering WebBinding from IIS"
        Remove-Item -path IIS:\SslBindings\0.0.0.0!443
      }

      # Download certificate from S3
      Read-S3Object -BucketName $bucket -Key $certkey -File $certfile

      # Install certificate
      Echo "Installing cert..."
      $securepwd = ConvertTo-SecureString -String $pwd -Force -AsPlainText
      $cert = Import-PfxCertificate -FilePath $certfile cert:\localMachine\my -Password
 $securepwd

      # Create site binding
      Echo "Creating and registering WebBinding"
      New-WebBinding -Name "Default Web Site" -IP "*" -Port 443 -Protocol https
      New-Item -path IIS:\SslBindings\0.0.0.0!443 -value $cert -Force

      ## Remove the HTTP binding
      ## (optional) Uncomment the following line to unbind port 80
      # Remove-WebBinding -Name "Default Web Site" -BindingInformation *:80:
      ##

      # Update firewall
      netsh advfirewall firewall add rule name="Open port 443" protocol=TCP
 localport=443 action=allow dir=OUT

commands:
```

```
  00_install_ssl:
    command: powershell -NoProfile -ExecutionPolicy Bypass -file C:\\certs\\install-
cert.ps1
```

単一インスタンスの環境では、インスタンスのセキュリティも変更してポート 443 のトラフィックを許可する必要があります。次の設定ファイルは、 AWS CloudFormation 関数を使用してセキュリティグループの ID を取得し、それにルールを追加します。

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

ロードバランシング環境では、エンドツーエンドの暗号化のために安全なトラフィックを変更なしでパスするか復号および暗号化することができるようにロードバランサーを設定します。

負荷分散された Elastic Beanstalk 環境でエンドツーエンドの暗号化を設定する

ロードバランサーでセキュアな接続を終了し、バックエンドへの接続には HTTP を使用することで十分なアプリケーションもあります。 AWS リソース間のネットワークトラフィックは、同じアカウントで実行中であっても、接続の一部ではないインスタンスがリッスンすることはできません。

ただし、厳格な外部規制を遵守する必要があるアプリケーションを開発している場合は、すべてのネットワーク接続をセキュリティで保護しなければならないことがあります。Elastic Beanstalk コンソールまたは設定ファイルを使用し、Elastic Beanstalk 環境のロードバランサーをバックエンドインスタンスに接続することで、これらの要件を満たすことができます。次の手順では、設定ファイルを使用します。

まだ追加していない場合は、まず、ロードバランサーにセキュアリスナーを追加します。

環境にインスタンスを設定して、安全なポートでリッスンして HTTPS 接続を終端する必要もあります。設定はプラットフォームによって異なります。手順については「インスタンスでの HTTPS 終端の設定」を参照してください。自己署名証明書を EC2 インスタンスで問題なく使用できます。

次に、アプリケーションが使用するセキュリティポートで HTTPS を使用してトラフィックを転送するように、リスナーを設定します。環境で使用するロードバランサーのタイプに基づいて、次のいずれかの設定ファイルを使用します。

.ebextensions/https-reencrypt-clb.config

この設定ファイルは、Classic Load Balancer で使用します。ロードバランサーの設定に加えて、設定ファイルは、ポート 443 および HTTPS を使用するようにデフォルトのヘルスチェックを変更して、ロードバランサーが安全に接続できることを確実にします。

```
option_settings:
  aws:elb:listener:443:
    InstancePort: 443
    InstanceProtocol: HTTPS
  aws:elasticbeanstalk:application:
    Application Healthcheck URL: HTTPS:443/
```

.ebextensions/https-reencrypt-alb.config

この設定ファイルは、Application Load Balancer で使用します。

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
    Protocol: HTTPS
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
    Protocol: HTTPS
```

.ebextensions/https-reencrypt-nlb.config

この設定ファイルは、Network Load Balancer で使用します。

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
```

`DefaultProcess`オプションの名前は、Application Load Balancer が特定のパスへのトラフィックについて同じポートにデフォルトでないリスナーを設定できることにちなんでいます (詳細については、「Application Load Balancer」を参照してください)。Network Load Balancer の場合、このオプションでは、このリスナーの唯一のターゲットプロセスを指定します。

この例では、セキュアな (HTTPS) トラフィックをリッスンするため、プロセスは `https` という名前になります。Network Load Balancer は TCP でのみ使用できるため、リスナーは、TCP プロトコルを使用して、指定ポートのプロセスにトラフィックを送信します。HTTP および HTTPS のネットワークトラフィックは TCP の上で実装されるため、これは問題ありません。

> **Note**
>
> EB CLI および Elastic Beanstalk コンソールでは、上記のオプションに推奨値が適用されます。設定ファイルを使用して同じファイルを設定する場合は、これらの設定を削除する必要があります。詳細については、「推奨値」を参照してください。

次のタスクでは、トラフィックを許可するようにロードバランサーのセキュリティグループを変更する必要があります。環境を起動する Amazon Virtual Private Cloud (Amazon VPC) (デフォルトの VPC またはカスタム VPC) によって、ロードバランサーのセキュリティグループが異なります。デフォルト VPC では、Elastic Load Balancing は、すべてのロードバランサーで使用できるデフォルトのセキュリティグループを提供します。カスタムの Amazon VPC では、ロードバランサーで使用するセキュリティグループが Elastic Beanstalk によって作成されます。

両方のシナリオをサポートするには、セキュリティグループを作成し、それを使用するように Elastic Beanstalk に指示することができます。次の設定ファイルは、セキュリティグループを作成して、ロードバランサーにアタッチします。

`.ebextensions/https-lbsecuritygroup.config`

```
option_settings:
  # Use the custom security group for the load balancer
  aws:elb:loadbalancer:
    SecurityGroups: '`{ "Ref" : "loadbalancersg" }`'
    ManagedSecurityGroup: '`{ "Ref" : "loadbalancersg" }`'

Resources:
  loadbalancersg:
    Type: AWS::EC2::SecurityGroup
    Properties:
```

```
      GroupDescription: load balancer security group
      VpcId: vpc-########
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 443
          ToPort: 443
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
      SecurityGroupEgress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
```

強調表示されたテキストをデフォルトまたはカスタム VPC ID に置き換えます。上記の例には、送受信にポート 80 を介して HTTP 接続を許可する設定が含まれています。安全な接続のみを許可する場合には、これらのプロパティを削除できます。

最後に、ロードバランサーのセキュリティグループとインスタンスのセキュリティグループ間でポート 443 経由の通信を許可する送受信ルールを追加します。

.ebextensions/https-backendsecurity.config

```
Resources:
  # Add 443-inbound to instance security group (AWSEBSecurityGroup)
  httpsFromLoadBalancerSG:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      SourceSecurityGroupId: {"Fn::GetAtt" : ["loadbalancersg", "GroupId"]}
  # Add 443-outbound to load balancer security group (loadbalancersg)
  httpsToBackendInstances:
    Type: AWS::EC2::SecurityGroupEgress
    Properties:
      GroupId: {"Fn::GetAtt" : ["loadbalancersg", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
```

```
      FromPort: 443
      DestinationSecurityGroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
```

これをセキュリティグループの作成とは別に行うことで、循環依存を生じさせることなく、送信元および送信先のセキュリティグループを制限することができます。

上記のすべてのタスクを完了すると、ロードバランサーは HTTPS を使用してバックエンドインスタンスに安全に接続します。ロードバランサーは、インスタンスの証明書が信頼できる認証局によって発行されたものであるかどうかを確認しません。また、提供される証明書をすべて受け入れます。

特定の証明書のみを信頼するように指示するポリシーをロードバランサーに追加することで、この動作を変更できます。以下の設定ファイルは 2 つのポリシーを作成します。1 つのポリシーは公開証明書を指定し、もう 1 つは、インスタンスポート 443 への接続用の証明書のみを信頼するようにロードバランサーに指示します。

.ebextensions/https-backendauth.config

```
option_settings:
  # Backend Encryption Policy
  aws:elb:policies:backendencryption:
    PublicKeyPolicyNames: backendkey
    InstancePorts:  443
  # Public Key Policy
  aws:elb:policies:backendkey:
    PublicKey: |
      -----BEGIN CERTIFICATE-----
      ################################################################
      ################################################################
      ################################################################
      ################################################################
      ################################################
      -----END CERTIFICATE-----
```

強調表示されたテキストを EC2 インスタンスの公開証明書の内容に置き換えます。

環境のロードバランサーを TCP パススルー用に設定する

AWS Elastic Beanstalk 環境内のロードバランサーで HTTPS トラフィックを復号しない場合は、バックエンドインスタンスにリクエストをそのまま中継するようにセキュアリスナーを設定できます。

> **Important**
>
> 復号せずに HTTPS トラフィックを中継するようにロードバランサーを設定することには、欠点があります。ロードバランサーが暗号化されたリクエストを見ることができないため、ルーティングを最適化したり、応答メトリクスをレポートしたりできません。

最初に環境の EC2 インスタンスが HTTPS を終了するように設定します。単一インスタンスの環境の設定をテストして、組み合わせにロードバランサーを追加する前に、すべてが機能していることを確認します。

設定ファイルをプロジェクトに追加して、TCP パケットをそのままバックエンドインスタンスのポート 443 に渡すように、ポート 443 のリスナーを設定します。

.ebextensions/https-lb-passthrough.config

```
option_settings:
  aws:elb:listener:443:
    ListenerProtocol: TCP
    InstancePort: 443
    InstanceProtocol: TCP
```

デフォルトの Amazon Virtual Private Cloud (Amazon VPC) では、インスタンスのセキュリティグループにルールを追加して、ロードバランサーから 443 への着信トラフィックを許可する操作も必要です。

.ebextensions/https-instance-securitygroup.config

```
Resources:
  443inboundfromloadbalancer:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      SourceSecurityGroupName: { "Fn::GetAtt": ["AWSEBLoadBalancer",
 "SourceSecurityGroup.GroupName"] }
```

カスタム VPC では、セキュリティグループ設定が Elastic Beanstalk によって更新されます。

HTTP から HTTPS へのリダイレクトの設定

このトピックでは、アプリケーションへの HTTP トラフィックをエンドユーザーが開始した場合でも、処理する方法について説明します。これを行うには、HTTP から HTTPS へのリダイレクトを設定します。これは、HTTPS の強制と呼ばれることもあります。

リダイレクトを設定するには、まず、HTTPS トラフィックを処理するように環境を設定します。次に、HTTP トラフィックを HTTPS にリダイレクトします。これらの 2 つのステップについては、以下のサブセクションで説明します。

HTTPS トラフィックを処理するように環境を設定する

環境の負荷分散設定に応じて、以下のいずれかの操作を行います。

- 負荷分散された環境 – HTTPS を終了するようにロードバランサーを設定します。
- 単一インスタンス環境 – インスタンスで HTTPS 接続を終了するようにアプリケーションを設定します。この設定は、環境のプラットフォームによって異なります。

HTTP トラフィックを HTTPS にリダイレクトする

アプリケーションの HTTP トラフィックを HTTPS にリダイレクトするには、環境のインスタンスのウェブサーバーまたは環境の Application Load Balancer を設定します。

インスタンスウェブサーバーを設定する

この方法は、任意のウェブサーバー環境で機能します。HTTP リダイレクトレスポンスステータスで HTTP トラフィックに応答するように、Amazon EC2 インスタンスのウェブサーバーを設定します。

この設定は、環境のプラットフォームによって異なります。GitHub の `https-redirect` コレクションでプラットフォームのフォルダを探して、そのフォルダ内のサンプル設定ファイルを使用します。

環境で Elastic Load Balancing ヘルスチェックを使用している場合、ロードバランサーは、正常なインスタンスが HTTP ヘルスチェックメッセージに HTTP 200 (OK) で応答することを想定します。したがって、ウェブサーバーはこれらのメッセージを HTTPS にリダイレクトすべきではありません。`https-redirect` のサンプル設定ファイルでは、この要件が正しく処理されています。

ロードバランサーを設定する

この方法は、Application Load Balancer を使用するロードバランスされた環境がある場合に機能します。Application Load Balancer は、HTTP トラフィックの受信時にリダイレクトレスポンスを送信できます。この場合、環境のインスタンスでリダイレクトを設定する必要はありません。

GitHub には、リダイレクト用に Application Load Balancer を設定する方法を示す 2 つのサンプル設定ファイルがあります。

- `alb-http-to-https-redirection-full.config` 設定ファイルでは、ポート 443 で HTTPS リスナーを作成し、受信 HTTP トラフィックを HTTPS にリダイレクトするようにデフォルトのポート 80 リスナーを変更します。
- `alb-http-to-https-redirection.config` 設定ファイルでは、443 リスナーが定義されることが想定されています。これを定義するには、標準の Elastic Beanstalk 設定名前空間、または Elastic Beanstalk コンソールを使用できます。次に、ポート 80 リスナーをリダイレクト用に変更します。

Elastic Beanstalk プラットフォーム

AWS Elastic Beanstalk には、アプリケーションを構築できるさまざまなプラットフォームが用意されています。ウェブアプリケーションをこれらのプラットフォームのいずれかに設計し、Elastic Beanstalk は選択したプラットフォームバージョンにコードをデプロイして、アクティブなアプリケーション環境を作成します。

Elastic Beanstalk は、さまざまなプログラミング言語、アプリケーションサーバー、および Docker コンテナのプラットフォームを提供します。プラットフォームによっては、同時にサポートされる複数のバージョンがあります。

トピック

- Elastic Beanstalk プラットフォームの用語集
- Elastic Beanstalk プラットフォームメンテナンスの責任共有モデル
- Elastic Beanstalk プラットフォームのサポートポリシー
- Elastic Beanstalk プラットフォームのリリーススケジュール
- Elastic Beanstalk でサポートされているプラットフォーム
- Elastic Beanstalk Linux プラットフォーム
- Elastic Beanstalk Linux プラットフォームの拡張

Elastic Beanstalk プラットフォームの用語集

以下は、 AWS Elastic Beanstalk プラットフォームとそのライフサイクルに関連する重要な用語です。

ランタイム

アプリケーションコードを実行するために必要なプログラミング言語固有のランタイムソフトウェア (フレームワーク、ライブラリ、インタプリタ、vm など)。

Elastic Beanstalk コンポーネント

Elastic Beanstalk がプラットフォームに追加して Elastic Beanstalk 機能を有効にするためのソフトウェアコンポーネント。たとえば、ヘルス情報を収集して報告するためは、拡張ヘルスエージェントが必要です。

プラットフォーム

オペレーティングシステム (OS)、ランタイム、ウェブサーバー、アプリケーションサーバー、および Elastic Beanstalk コンポーネントの組み合わせです。プラットフォームは、アプリケーションを実行するために使用できるコンポーネントを提供します。

プラットフォームのバージョン

特定のバージョンのオペレーティングシステム (OS)、ランタイム、ウェブサーバー、アプリケーションサーバー、および Elastic Beanstalk コンポーネントの組み合わせです。プラットフォームのバージョンに基づいて Elastic Beanstalk 環境を作成し、その環境にアプリケーションをデプロイします。

プラットフォームのバージョンには、X.Y.Z 形式のセマンティックバージョン番号があります。ここで X はメジャーバージョン、Y はマイナーバージョン、Z はパッチバージョンをそれぞれ表します。

プラットフォームのバージョンは、次のいずれかの状態になります。

- 推奨 — サポートされているプラットフォームブランチの最新のプラットフォームバージョン。このバージョンにはup-to-dateコンポーネントが含まれており、本番環境での使用をお勧めします。Elastic Beanstalk が新しいプラットフォームバージョンをリリースすると、新しいバージョンは以前のバージョンより優先され、対応するプラットフォームブランチの推奨プラットフォームバージョンになります。
- 非推奨 – プラットフォームブランチの最新バージョンではないプラットフォームバージョン。これらのバージョンは引き続き機能する可能性がありますが、最新のプラットフォームバージョンに更新することを強くお勧めします。マネージドプラットフォームの更新を使用して、を自動的にup-to-date状態に保つことができます。

AWS CLI コマンドdescribe-platform-versionを使用して `PlatformLifecycleState`フィールドをチェックすることで、プラットフォームバージョンが推奨されているかどうかを確認できます。

プラットフォームブランチ

オペレーティングシステム (OS)、ランタイム、Elastic Beanstalk コンポーネントなど、一部のコンポーネントの特定バージョン (通常はメジャー) を共有する一連のプラットフォームバージョンです。例: 64 ビット Amazon Linux 2023 で実行される Python 3.13、64 ビット Windows Server 2025 で実行される IIS 10.0。プラットフォームブランチは、新しいプラットフォームバージョンの形式で更新を受け取ります。ブランチ内の後続の各プラットフォームバージョンは、前のバージョンへの更新です。

サポートされている各プラットフォームブランチの推奨バージョンは、環境の作成に無条件に使用できます。以前のプラットフォームバージョンは、新しいバージョンに置き換えられた時点で使用していた環境がある場合、引き続き使用できます。これには、アクティブな環境と、その後終了した可能性のある最近作成された環境の両方が含まれます。以前のプラットフォームバージョンにはup-to-dateコンポーネントがないため、使用はお勧めしません。

プラットフォームブランチは、次のいずれかの状態になります。

- サポート対象 – 現在のプラットフォームブランチ。これは、全体がサポートされているコンポーネントで構成されています。サポートされているコンポーネントは、サプライヤーが指定したサポート終了 (EOL) に達していません。継続的なプラットフォーム更新を受け取り、実稼働環境での使用を推奨します。サポートされているプラットフォームブランチのリストについては、「AWS Elastic Beanstalk プラットフォーム」ガイドの「Elastic Beanstalk でサポートされているプラットフォーム」を参照してください。
- ベータ – プレビュー、プレリリースのプラットフォームブランチ。それは本質的に実験的なものです。しばらくの間、継続的なプラットフォームの更新を受け取る可能性がありますが、長期的なサポートはありません。ベータ版のプラットフォームブランチは、実稼働環境での使用にはお勧めしません。評価のためにのみ使用してください。ベータプラットフォームブランチのリストについては、「AWS Elastic Beanstalk プラットフォーム」ガイドの「パブリックベータの Elastic Beanstalk プラットフォームのバージョン」を参照してください。
- 非推奨 – 1 つ以上のコンポーネント (ランタイムやオペレーティングシステムなど) がサポート終了 (EOL) に近づいているか、サプライヤーが指定した EOL に達したプラットフォームブランチ。廃止されたプラットフォームブランチは、廃止日まで新しいプラットフォームバージョンを受信し続けますが、EOL に達したコンポーネントは更新を受信しません。たとえば、ランタイムバージョンが EOL に達すると、プラットフォームブランチは非推奨としてマークされますが、プラットフォームブランチの廃止日までオペレーティングシステムの更新が引き続き受信されます。プラットフォームブランチは、引き続き EOL ランタイムバージョンの更新を受信しません。廃止されたプラットフォームブランチの使用はお勧めしません。
- 廃止 – 更新を受け取らなくなったプラットフォームブランチ。廃止されたプラットフォームブランチは、Elastic Beanstalk コンソールを使用して新しい Elastic Beanstalk 環境を作成することはできません。環境が廃止されたプラットフォームブランチを使用している場合は、サポートされているプラットフォームブランチに更新して、更新を引き続き受信する必要があります。廃止されたプラットフォームブランチの使用はお勧めしません。廃止されたプラットフォームブランチの詳細については、「」を参照してくださいthe section called “プラットフォームのサポートポリシー”。リタイアが予定されているプラットフォームブランチのリストについては、「リタイアするプラットフォームブランチのスケジュール」を参照してくだ

さい。過去に廃止されたプラットフォームブランチを確認するには、「廃止されたプラットフォームブランチ履歴」を参照してください。

環境が非推奨またはリタイアしたプラットフォームブランチを使用している場合は、サポートされているプラットフォームブランチのプラットフォームバージョンに更新することをお勧めします。詳細については、「the section called “プラットフォームの更新”」を参照してください。

AWS CLI コマンドを使用して `PlatformBranchLifecycleState`フィールドをチェックdescribe-platform-versionすることで、プラットフォームブランチの状態を確認できます。

プラットフォームの更新

OS、ランタイム、ウェブサーバー、アプリケーションサーバー、Elastic Beanstalk コンポーネントなど、プラットフォームの一部のコンポーネントの更新を含む新しいプラットフォームバージョンのリリース。Elastic Beanstalk が新しいプラットフォームバージョンをリリースすると、新しいバージョンは以前のバージョンより優先され、対応するプラットフォームブランチの推奨プラットフォームバージョンになります。プラットフォームの更新はセマンティックバージョンの分類に従い、次の 3 つのレベルを設定できます。

- 主な更新 – 既存のプラットフォームバージョンと互換性がない変更を含む更新。新しいメジャーバージョンで正しく実行されるようにアプリケーションを変更する必要がある場合があります。主な更新には、新しいプラットフォームのメジャーバージョン番号があります。
- マイナー更新 – ほとんどの場合、既存のプラットフォームバージョンと下位互換性がある変更を含む更新。アプリケーションによっては、新しいマイナーバージョンで正しく実行されるようにアプリケーションを変更する必要がある場合があります。マイナーな更新には、新しいプラットフォームのマイナーバージョン番号があります。
- パッチ更新 – 既存のプラットフォームのバージョンとの下位互換性がある、メンテナンスリリース (バグ修正、セキュリティの更新、パフォーマンスの改善) で構成される更新。パッチ更新には、新しいパッチのプラットフォームバージョン番号があります。

管理された更新

Elastic Beanstalk でサポートされているプラットフォームバージョンのオペレーティングシステム (OS)、ランタイム、ウェブサーバー、アプリケーションサーバー、および Elastic Beanstalk コンポーネントにパッチおよびマイナーな更新を自動的に適用する Elastic Beanstalk の機能。管理された更新は、同じプラットフォームブランチの新しいプラットフォームバージョンを環境に適用します。管理された更新がパッチ更新のみ、またはマイナーおよびパッチ更新に適用されるよう設定できます。管理された更新を完全に無効にすることもできます。

詳細については、「マネージドプラットフォーム更新」を参照してください。

Elastic Beanstalk プラットフォームメンテナンスの責任共有モデル

AWS とお客様は、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の高レベルを達成する責任を共有します。この責任共有モデルにより、運用上の負担が軽減されます。

詳細については、 AWS 「責任共有モデル」を参照してください。

AWS Elastic Beanstalk は、マネージド更新機能を提供することで、責任共有モデルのユーザー側を実行するのに役立ちます。この機能では、Elastic Beanstalk でサポートされているプラットフォームのバージョンに対してパッチとマイナーな更新を自動的に適用します。マネージド型の更新が失敗した場合、お客様がそれを確実に認識してすぐにアクションを実行できるように、Elastic Beanstalk は失敗について通知します。

詳細については、「マネージドプラットフォーム更新」を参照してください。

さらに、 Elastic Beanstalk は次のことを行います。

- 今後 12 か月のそのプラットフォームのサポートポリシーとリタイアのスケジュールを発行します。
- オペレーティングシステム (OS)、ランタイム、アプリケーションサーバー、ウェブサーバーコンポーネントのパッチ、マイナーな更新、およびメジャーな更新を、通常は利用可能になってから 30 日以内にリリースします。Elastic Beanstalk は、サポートされているプラットフォームのバージョンに含まれている Elastic Beanstalk コンポーネントの更新を作成します。その他のすべての更新は、サプライヤ (所有者またはコミュニティ) から直接取得されます。

サポート対象プラットフォームのすべてのアップデートは、AWS Elastic Beanstalk リリースノートガイドの リリースノート でお知らせします。また、AWS Elastic Beanstalk プラットフォームガイドには、サポートされているすべてのプラットフォームとそのコンポーネントのリスト、プラットフォームの履歴が記載されています。詳細については、「サポートされているプラットフォームとコンポーネント履歴」を参照してください。

お客様は次の操作を行う必要があります。

- 管理するすべてのコンポーネントを更新します (責任共有モデルでは顧客として識別されます)。AWS https://aws.amazon.com/compliance/shared-responsibility-model/これには、アプリケーションのセキュリティ、データ、アプリケーションに必要でお客様がダウンロードしたコンポーネントのセキュリティの確保が含まれます。

- Elastic Beanstalk 環境が、サポートされているプラットフォームバージョンで実行されていることを確認します。リタイアしているプラットフォームバージョンで実行されている環境は、サポートされているバージョンに移行します。
- Elastic Beanstalk 環境にカスタム Amazon マシンイメージ (AMI) を使用している場合は、カスタム AMI へのパッチ適用、メンテナンス、およびテストによって、サポートされている Elastic Beanstalk プラットフォームバージョンと互換性がある最新の状態を維持します。カスタム AMI での環境管理の詳細については、「Elastic Beanstalk 環境でのカスタム Amazon マシンイメージ (AMI) の使用」を参照してください。
- 管理された更新で失敗したすべての問題を解決し、更新を再試行します。
- Elastic Beanstalk で管理されている更新をオプトアウトした場合は、OS、ランタイム、アプリケーションサーバー、ウェブサーバーに自分でパッチを適用します。これを行うには、手動でプラットフォームの更新を適用するか、関連するすべての環境リソースでコンポーネントに直接パッチを適用します。
- 責任 AWS 共有モデルに従って、Elastic Beanstalk の外部で使用する AWS サービスのセキュリティとコンプライアンスを管理します。

Elastic Beanstalk プラットフォームのサポートポリシー

Elastic Beanstalk は、サプライヤー (所有者またはコミュニティ) から引き続き継続的なマイナーおよびパッチ更新を受け取るプラットフォームブランチをサポートします。関連用語の詳細な定義については、「Elastic Beanstalk プラットフォームの用語集」を参照してください。

廃止されたプラットフォームブランチ

サポートされているプラットフォームブランチのコンポーネントがサプライヤーによって End of Life (EOL) とマークされた場合、Elastic Beanstalk はそのプラットフォームブランチを廃止とマークします。プラットフォームブランチのコンポーネントには、オペレーティングシステム (OS)、ランタイム言語バージョン、アプリケーションサーバー、ウェブサーバーなどがあります。

プラットフォームブランチが廃止とマークされると、次のポリシーが適用されます。

- Elastic Beanstalk は、セキュリティ更新を含むメンテナンス更新の提供を停止します。
- Elastic Beanstalk では、廃止されたプラットフォームブランチのテクニカルサポートがなくなります。
- Elastic Beanstalk は、このプラットフォームブランチを、Elastic Beanstalk の新しいお客様が新しい環境へのデプロイに使用できるとマークしなくなります。リタイアしたプラットフォームブラン

チで実行されているアクティブな環境を持つ既存のお客様には、発表されたリタイア日から 90 日間の猶予期間があります。

> **Note**
>
> 廃止されたプラットフォームブランチは、Elastic Beanstalk コンソールでは使用できません。ただし、廃止されたプラットフォームブランチに基づく既存の環境を持つお客様は AWS CLI、EB CLI および EB API を通じて利用できます。また、既存のお客様は [環境のクローンを作成] コンソールと [環境の再構築] コンソールを使用できます。

廃止が予定されているプラットフォームブランチのリストについては、次の「Elastic Beanstalk プラットフォームスケジュール」トピックの「廃止されるプラットフォームブランチのスケジュール」を参照してください。

環境のプラットフォームブランチの廃止時に予想されることの詳細については、「プラットフォームの廃止に関するよくある質問」を参照してください。

90 日間の猶予期間経過後

廃止されたプラットフォームブランチのポリシーでは、環境へのアクセスを削除したり、リソースを削除したりすることはありません。ただし、廃止されたプラットフォームブランチで Elastic Beanstalk 環境を実行することにはリスクがあるため、既存のお客様は注意する必要があります。そのような環境は、予測不能な状況になることがあります。サプライヤーがコンポーネントを EOL とマークしているので、廃止されたプラットフォームブランチに対して Elastic Beanstalk でセキュリティ更新プログラム、テクニカルサポート、修正プログラムを提供できないためです。

例えば、リタイアしたプラットフォームブランチで実行されている環境に、有害で重大なセキュリティ脆弱性が現れる可能性があります。または、時間の経過とともに Elastic Beanstalk サービスとの互換性がなくなると、環境での EB API アクションの動作が停止することがあります。このようなリスクの可能性は、リタイアしたプラットフォームブランチ上の環境が長くアクティブなままであるほど、増加します。最近のリリースでコンポーネントサプライヤーから提供される重要なセキュリティ、パフォーマンス、および機能拡張のメリットを継続的に利用できるよう、すべての Elastic Beanstalk 環境を、サポートされているプラットフォームバージョンに更新することを強くお勧めします。

廃止されたプラットフォームブランチでの実行中にアプリケーションに問題が発生し、サポートされているプラットフォームに移行できない場合は、他の代替方法を検討する必要があります。回避策と

して、アプリケーションを Docker イメージにカプセル化して Docker コンテナとして実行する方法があります。これにより、お客様は、Elastic Beanstalk AL2023/AL2 Docker プラットフォームなどの当社の Docker ソリューションや、Amazon ECS や Amazon EKS などのその他の Docker ベースのサービスを使用できるようになります。Docker 以外の代替方法には、必要なランタイムを完全にカスタマイズできる AWS CodeDeploy サービスが含まれます。

Elastic Beanstalk プラットフォームのリリーススケジュール

新しいプラットフォームブランチバージョンの毎月の定期的なリリースに加えて、リリースメンテナンスには以下のプロセスも含まれています。

- 新しいプラットフォームブランチのリリース – これは通常、ランタイム言語、オペレーティングシステム、またはアプリケーションサーバーの新しいメジャーバージョンの導入です。
- プラットフォームブランチの廃止 – いずれかのコンポーネントが End of Life (EOL) を迎えたときに、プラットフォームブランチを廃止する必要があります。廃止されるブランチのポリシーの詳細については、「Elastic Beanstalk プラットフォームのサポートポリシー」を参照してください。

トピック

- リソースを計画する
- 今後のプラットフォームブランチリリース
- 廃止されるプラットフォームブランチのスケジュール
- リタイアしたプラットフォームブランチの履歴
- 廃止されたサーバーとオペレーティングシステムの履歴

リソースを計画する

以下のリソースは、Elastic Beanstalk プラットフォームで実行されているアプリケーションのメンテナンスとサポートを計画するのに役立ちます。

- AWS Elastic Beanstalk プラットフォームガイド — このガイドでは、各プラットフォームブランチの詳細なコンポーネントリストを提供します。また、リリース日ごとに同じ詳細な内容を含むプラットフォーム履歴も記載されています。このガイドで、プラットフォームブランチの特定のコンポーネントがいつ変更されたかがわかります。アプリケーションの動作が変わった場合は、プラットフォームガイドで発生日を相互参照して、アプリケーションに影響を与えた可能性のあるプラットフォームの変更があったかどうかを確認することもできます。

- AWS Elastic Beanstalk リリースノート — リリースノートは、マイナーリリースとメジャーリリースの両方を含むすべてのプラットフォームリリースを発表しました。これには、毎月のプラットフォーム更新、セキュリティリリース、修正プログラム、廃止のお知らせが含まれます。リリースノートのドキュメントから RSS フィードをサブスクライブできます。

今後のプラットフォームブランチリリース

次の表は、今後の Elastic Beanstalk プラットフォームブランチとその目標リリース日の一覧です。これらの日付は暫定的なものであり、変更される可能性があります。

ランタイムバージョン/プラットフォームブランチ	オペレーティングシステム	目標リリース日
Corretto 25	Amazon Linux 2023	2025 年 10 月
Corretto 25 with Tomcat 11	Amazon Linux 2023	2025 年 10 月
Python 3.14	Amazon Linux 2023	2025 年 11 月
Node.js 24	Amazon Linux 2023	2025 年 11 月
.NET 10	Amazon Linux 2023	2025 年 12 月
PHP 8.5	Amazon Linux 2023	2026 年 1 月
Ruby 3.5	Amazon Linux 2023	2026 年 2 月

廃止されるプラットフォームブランチのスケジュール

次の表は、一部のコンポーネントが End of Life (EOL) を迎えるため廃止が予定されている Elastic Beanstalk プラットフォームブランチの一覧です。すべての AL2-basedプラットフォームブランチには、Amazon Linux 2 が EOL に達する 2026 年 6 月 30 日までに廃止日があります。Amazon Linux 2 の詳細については、「Amazon Linux 2 のFAQs」を参照してください。

特定のコンポーネントを含むため廃止されるプラットフォームブランチの詳細リストについては、「AWS Elastic Beanstalk プラットフォーム」ガイドの廃止されるプラットフォームバージョンに関するページを参照してください。

ランタイムバージョン/プラットフォームブランチ	廃止予定日	
PHP 8.1 AL2023	2026 年 3 月 31 日	
PHP 8.1 AL2	2026 年 3 月 31 日	
Docker AL2	2026 年 6 月 30 日	
ECS AL2	2026 年 6 月 30 日	
Go 1 AL2	2026 年 6 月 30 日	
Corretto 8 AL2	2026 年 6 月 30 日	
Corretto 11 AL2	2026 年 6 月 30 日	
Corretto 17 AL2	2026 年 6 月 30 日	
Corretto 8 with Tomcat 9 AL2	2026 年 6 月 30 日	
Corretto 11 with Tomcat 9 AL2	2026 年 6 月 30 日	
.NET Core AL2	2026 年 6 月 30 日	
Python 3.9 AL2023	2026 年 7 月 31 日	
Ruby 3.2 AL2023	2026 年 7 月 31 日	
Node.js 20 AL2023	2026 年 7 月 31 日	
.NET 9 AL2023	2026 年 7 月 31 日	
IIS 10.0 on Windows Server 2016 (& Core)	2026 年 9 月 30 日	

リタイアしたプラットフォームブランチの履歴

次の表は、既に廃止済の Elastic Beanstalk プラットフォームブランチの一覧です。これらのプラットフォームブランチとそのコンポーネントの詳細な履歴は、「AWS Elastic Beanstalk プラットフォーム」ガイドの「プラットフォーム履歴」で確認できます。

Amazon Linux 2023 (AL2023)

ランタイムバージョン/プラットフォームブランチ	リタイア日		
.NET 6 AL2023	2025 年 4 月 8 日		
Node.js 18 AL2023	2025 年 8 月 11 日		

Amazon Linux 2 (AL2)

ランタイムバージョン/プラットフォームブランチ	リタイア日		
Corretto 11 with Tomcat 8.5 AL2	2024 年 10 月 10 日		
Corretto 8 with Tomcat 8.5 AL2	2024 年 10 月 10 日		
Corretto 11 with Tomcat 7 AL2	2022 年 6 月 29 日		
Corretto 8 with Tomcat 7 AL2	2022 年 1 月 29 日		
Node.js 18 AL2	2025 年 8 月 11 日		

ランタイムバージョン/プラットフォームブランチ	リタイア日		
Node.js 16 AL2	2024 年 10 月 10 日		
Node.js 14 AL2	2024 年 10 月 10 日		
Node.js 12 AL2	2022 年 12 月 23 日		
Node.js 10 AL2	2022 年 1 月 29 日		
PHP 8.0 AL2	2024 年 10 月 10 日		
PHP 7.4 AL2	2023 年 6 月 9 日		
PHP 7.3 AL2	2022 年 6 月 29 日		
PHP 7.2 AL2	2022 年 1 月 29 日		
Python 3.8 AL2	2025 年 4 月 8 日		
Python 3.7 AL2	2024 年 10 月 10 日		
Ruby 3.0 AL2	2024 年 10 月 10 日		
Ruby 2.7 AL2	2024 年 10 月 10 日		
Ruby 2.6 AL2	2022 年 12 月 23 日		
Ruby 2.5 AL2	2022 年 1 月 29 日		

Amazon Linux AMI (AL1)

ランタイムバージョン/プラットフォームブランチ	リタイア日		
Single Container Docker	2022 年 7 月 18 日		
Multicontainer Docker	2022 年 7 月 18 日		
Preconfigured Docker - GlassFish 5.0 with Java 8	2022 年 7 月 18 日		
Go 1	2022 年 7 月 18 日		
Java 8	2022 年 7 月 18 日		
Java 7	2022 年 7 月 18 日		
Java 8 with Tomcat 8.5	2022 年 7 月 18 日		
Java 7 with Tomcat 7	2022 年 7 月 18 日		
Node.js	2022 年 7 月 18 日		
PHP 7.2 - 7.3	2022 年 7 月 18 日		
Python 3.6	2022 年 7 月 18 日		
Ruby 2,4, 2.5, 2.6 with Passenger	2022 年 7 月 18 日		

ランタイムバージョン/プラットフォームブランチ	リタイア日		
Ruby 2.4, 2.5, 2.6 with Puma	2022 年 7 月 18 日		
Go 1.3–1.10	2020 年 10 月 31 日		
Java 6	2020 年 10 月 31 日		
Node.js 4.x–8.x	2020 年 10 月 31 日		
PHP 5.4–5.6	2020 年 10 月 31 日		
PHP 7.0–7.1	2020 年 10 月 31 日		
Python 2.6、2.7、3.4	2020 年 10 月 31 日		
Ruby 1.9.3	2020 年 10 月 31 日		
Ruby 2.0–2.3	2020 年 10 月 31 日		

> **Note**
>
> 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。詳細については、「プラットフォームの廃止に関するよくある質問」を参照してください。

Windows Server

ランタイムバージョン/プラットフォームブランチ	リタイア日		
64 ビット Windows Server (& Core) 2012 R2 で実行される IIS 8.5	2023 年 12 月 4 日		
64 ビット Windows Server (および Core) 2012 R2 バージョン 0.1.0 で動作する IIS 8.5	2022 年 6 月 29 日		
64 ビット Windows Server (および Core) 2012 R2 バージョン 1.2.0 で動作する IIS 8.5	2022 年 6 月 29 日		
64 ビット Windows Server 2016 (および Core) バージョン 1.2.0 で動作する IIS 10.0	2022 年 6 月 29 日		
64 ビット Windows Server 2012 R1 プラットフォームブラ	2022 年 6 月 22 日		

ランタイムバージョン/プラットフォームブランチ	リタイア日		
ンチで動作する IIS 8			
64 ビット Windows Server 2012 R1 バージョン 0.1.0 で動作する IIS 8	2022 年 6 月 22 日		
64 ビット Windows Server 2012 R1 バージョン 1.2.0 で動作する IIS 8	2022 年 6 月 22 日		

> **Note**
>
> Windows 2012 R2 プラットフォームブランチの廃止についての詳細は、「AWS Elastic Beanstalk リリースノート」の「Windows Server 2012 R2 プラットフォームブランチの廃止」を参照してください。

廃止されたサーバーとオペレーティングシステムの履歴

次の表は、Elastic Beanstalk プラットフォームでサポートされなくなったオペレーティングシステム、アプリケーションサーバー、ウェブサーバーの履歴を示しています。これらのコンポーネントを利用したすべてのプラットフォームブランチは廃止されています。日付は、コンポーネントを含む最後の Elastic Beanstalk プラットフォームブランチの廃止日を反映しています。

オペレーティングシステム

OS バージョン	プラットフォームの廃止日		
Windows Server 2012 R2 running IIS 8.5	2023 年 12 月 4 日		
Windows Server Core 2012 R2 running IIS 8.5	2023 年 12 月 4 日		
Amazon Linux AMI (AL1)	2022 年 7 月 18 日		
Windows Server 2012 R1	2022 年 6 月 22 日		
Windows サーバー 2008 R2	2019 年 10 月 28 日		

アプリケーションサーバー

アプリケーションサーバーのバージョン	プラットフォームの廃止日		
Tomcat 7	2022 年 6 月 29 日 Amazon Linux 2 (AL2) プラットフォーム向け 2022 年 6 月 18 日 Amazon Linux AMI (AL1) プラットフォーム向け		
Tomcat 8	2020 年 10 月 31 日		
Tomcat 6	2020 年 10 月 31 日		

ウェブサーバー

ウェブサーバーのバージョン	プラットフォームの廃止日		
64 ビット Windows Server で動作する IIS 8	2022 年 6 月 22 日		
Apache HTTP Server 2.2	2020 年 10 月 31 日		
Nginx 1.12.2	2020 年 10 月 31 日		

Elastic Beanstalk でサポートされているプラットフォーム

AWS Elastic Beanstalk には、アプリケーションを構築できるさまざまなプラットフォームが用意されています。ウェブアプリケーションをこれらのプラットフォームのいずれかに設計し、Elastic Beanstalk は選択したプラットフォームバージョンにコードをデプロイして、アクティブなアプリケーション環境を作成します。

Elastic Beanstalk は、1 つまたは複数の Amazon EC2 インスタンスなどのアプリケーションを実行するために必要なリソースをプロビジョニングします。Amazon EC2 インスタンスで実行されるソフトウェアスタックは、環境用に選択した特定のプラットフォームバージョンによって異なります。

プラットフォームブランチのソリューションスタック名

特定のプラットフォームブランチバージョンのソリューションスタック名を使用して、環境を EB CLI、Elastic Beanstalk API、または AWS CLI で起動できます。「AWS Elastic Beanstalk Platforms」ガイドには、「Elastic Beanstalk がサポートするプラットフォーム」セクションと「プラットフォーム履歴」セクションの両方で、プラットフォームブランチバージョンの下にソリューションスタック名が一覧表示されています。

環境の作成に使用できるすべてのソリューションスタック名を取得するには、ListAvailableSolutionStacks API または AWS CLI の `aws elasticbeanstalk list-available-solution-stacks` を使用します。

アプリケーションがプラットフォームで依存しているソフトウェアをカスタマイズおよび設定できます。詳細については、「Linux サーバーでのソフトウェアのカスタマイズ」および「Windows サー

バーでのソフトウェアのカスタマイズ」を参照してください。最近のリリースに関する詳細なリリースノートは、「AWS Elastic Beanstalk リリースノート」で参照できます。

サポートされているプラットフォームとコンポーネント履歴

「AWS Elastic Beanstalk プラットフォーム」ガイドの「Elastic Beanstalk でサポートされているプラットフォーム」セクションには、現在のすべてのプラットフォームブランチバージョンの一覧があります。「プラットフォーム」ガイドには、以前のブランチプラットフォームバージョンのリストを含む、各プラットフォームのプラットフォーム履歴の一覧もあります。各プラットフォームのプラットフォーム履歴を表示するには、次のいずれかのリンクを選択します。

- Docker
- Go
- Java SE
- Tomcat (Java SE を実行)
- .NET Core on Linux
- Windows Server の .NET
- Node.js
- PHP
- Python
- Ruby

Elastic Beanstalk Linux プラットフォーム

Elastic Beanstalk Linux プラットフォームには、多くの機能がすぐに使用できるように用意されています。アプリケーションをサポートするために、いくつかの方法でプラットフォームを拡張できます。詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

Elastic Beanstalk がサポートするプラットフォームのほとんどは、Linux オペレーティングシステムに基づいています。具体的には、これらのプラットフォームは、 が提供する Linux ディストリビューションである Amazon Linux に基づいています AWS。Elastic Beanstalk Linux プラットフォームは、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを使用し、それらのインスタンスは Amazon Linux を実行します。

トピック

- サポートされている Amazon Linux のバージョン
- Elastic Beanstalk Linux プラットフォームのリスト
- インスタンスデプロイワークフロー
- Amazon Linux 2 以降で動作する ECS のインスタンスデプロイのワークフロー
- Elastic Beanstalk 環境用のプラットフォームスクリプトツール

サポートされている Amazon Linux のバージョン

AWS Elastic Beanstalk は、Amazon Linux 2 および Amazon Linux 2023 に基づくプラットフォームをサポートしています。

Amazon Linux 2 および Amazon Linux 2023 の詳細については、次を参照してください。

- Amazon Linux 2 –「Amazon EC2 ユーザーガイド」の「Amazon Linux」。
- Amazon Linux 2023 –「Amazon Linux 2023 ユーザーガイド」の「Amazon Linux 2023 とは」

サポートされているプラットフォームバージョンの詳細については、「Elastic Beanstalk でサポートされているプラットフォーム」を参照してください。

> **Note**
>
> Elastic Beanstalk AL1 または AL2 プラットフォームブランチから同等の AL2023 プラットフォームブランチにアプリケーションを移行できます。詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

Amazon Linux 2023

AWS は、2023 年 3 月に Amazon Linux 2023 の一般提供開始を発表しました。「Amazon Linux 2023 ユーザーガイド」には、Amazon Linux 2 と Amazon Linux 2023 の主な違いがまとめられています。詳細については、「ユーザーガイド」の「Amazon Linux 2 と Amazon Linux 2023 の比較」を参照してください。

Elastic Beanstalk Amazon Linux 2 プラットフォームと Amazon Linux 2023 プラットフォームの間には高度な互換性があります。ただし、留意すべき違いがいくつかあります。

- インスタンスメタデータサービスバージョン 1 (IMDSv1) – AL2023 プラットフォームでは、DisableIMDSv1 オプション設定がデフォルトで `true` に設定されます。デフォルトは AL2 プラットフォーム上の `false` です。
- pkg-repo インスタンスツール – pkg-repo ツールは、AL2023 プラットフォームで実行されている環境では使用できません。ただし、パッケージとオペレーティングシステムの更新を AL2023 インスタンスに手動で適用することはできます。詳細については、「Amazon Linux 2023 ユーザーガイド」の「パッケージとオペレーティングシステムの更新の管理」を参照してください。
- Apache HTTPd 設定 – AL2023 プラットフォームの Apache `httpd.conf` ファイルには、AL2 の構成設定とは異なるいくつかの構成設定があります。
  - デフォルトでは、サーバーのファイルシステム全体へのアクセスを拒否します。これらの設定については、Apacheウェブサイトの「セキュリティのヒント」ページの「デフォルトでサーバーファイルを保護する」で説明されています。
  - 設定したセキュリティ機能をユーザーが上書きできないようにします。この設定では、特別に有効になっているディレクトリを除き、すべてのディレクトリの `.htaccess` の設定へのアクセスが拒否されます。この設定については、Apache ウェブサイトの「セキュリティのヒント」ページの「システム設定の保護」で説明されています。「Apache HTTP サーバーチュートリアル: .htaccess ファイル」ページには、この設定がパフォーマンスの改善に役立つ可能性がある旨が記載されています。
  - 名前パターン `.ht*` のファイルへのアクセスを拒否します。この設定により、ウェブクライアントは `.htaccess` および `.htpasswd` ファイルを表示できなくなります。

上記の構成設定は、ご使用の環境に合わせて変更できます。詳細については、「Apache HTTPD の設定」を参照してください。

Elastic Beanstalk Linux プラットフォームのリスト

次のリストは、さまざまなプログラミング言語と Docker コンテナに対して Elastic Beanstalk がサポートする Linux プラットフォームを示します。Elastic Beanstalk は、これらすべてに対して、Amazon Linux 2 および Amazon Linux 2023 に基づくプラットフォームを提供します。プラットフォームの詳細については、該当するリンクを選択してください。

- Docker (および ECS Docker)
- Go
- Tomcat (Java SE を実行)
- Java SE

- .NET Core on Linux
- Node.js
- PHP
- Python
- Ruby

インスタンスデプロイワークフロー

> **Note**
>
> このセクションの情報は、Amazon Linux 2 および Amazon Linux 2023 で動作する ECS プラットフォームブランチには適用されません。詳細については、次のセクション「Amazon Linux 2 以降で動作する ECS のインスタンスデプロイのワークフロー」をご覧ください。

環境のプラットフォームを拡張する方法が多数あるため、Elastic Beanstalk がインスタンスをプロビジョニングしたり、インスタンスへのデプロイを実行したりするたびに何が起こるかを知ることは有益です。次の図は、このデプロイワークフロー全体を示しています。デプロイのさまざまなフェーズと、各フェーズで Elastic Beanstalk が実行するステップを示します。

> **メモ**
>
> - この図は、デプロイ中に環境インスタンスで Elastic Beanstalk が実行する一連のステップ全体を表しているわけではありません。この図は、カスタマイズの実行順序とコンテキストを提供するために、説明のために提供します。
> - 簡単にするために、図では、`.platform/hooks/*` フックのサブディレクトリ (アプリケーションデプロイ用) のみを示しており、`.platform/confighooks/*` フックのサブディレクトリ (設定デプロイ用) は示していません。後者のサブディレクトリのフックは、図に示す対応するサブディレクトリのフックとまったく同じステップで実行されます。

デプロイフェーズとステップの詳細を次に示します。

1. 最初のステップ

Elastic Beanstalk は、アプリケーションをダウンロードして抽出します。これらの各ステップの後、Elastic Beanstalk は拡張性ステップの 1 つを実行します。

a. 設定ファイルの commands: セクションにあるコマンドを実行します。

b. ソースバンドルの `.platform/hooks/prebuild` ディレクトリにある実行可能ファイルを実行します (設定デプロイ用の `.platform/confighooks/prebuild`)。

2. 構成する

Elastic Beanstalk は、アプリケーションとプロキシサーバーを設定します。

a. ソースバンドルの `Buildfile` にあるコマンドを実行します。

b. ソースバンドルの `.platform/nginx` ディレクトリにカスタムプロキシ設定ファイルがある場合は、そのランタイムの場所にコピーします。

c. 設定ファイルの container_commands: セクションにあるコマンドを実行します。

d. ソースバンドルの `.platform/hooks/predeploy` ディレクトリにある実行可能ファイルを実行します (設定デプロイ用の `.platform/confighooks/predeploy`)。

3. デプロイ

Elastic Beanstalk は、アプリケーションとプロキシサーバーをデプロイして実行します。

a. ソースバンドルの `Procfile` ファイルにあるコマンドを実行します。

b. カスタムプロキシ設定ファイルがあれば、プロキシサーバーを実行または再実行します。

c. ソースバンドルの `.platform/hooks/postdeploy` ディレクトリにある実行可能ファイルを実行します (設定デプロイ用の `.platform/confighooks/postdeploy`)。

Amazon Linux 2 以降で動作する ECS のインスタンスデプロイのワークフロー

前のセクションでは、アプリケーションデプロイのワークフローのフェーズ全体でサポートされる拡張機能について説明します。Docker プラットフォームブランチ Amazon Linux 2 以降で動作する ECS にはいくつかの相違点があります。このセクションでは、これらの概念がこの特定のプラットフォームブランチにどのように適用されるかについて説明します。

環境のプラットフォームを拡張する方法が多数あるため、Elastic Beanstalk がインスタンスをプロビジョニングしたり、インスタンスへのデプロイを実行したりするたびに何が起こるかを知ることは有益です。次の図は、Amazon Linux 2 で動作する ECS および Amazon Linux 2023 で動作する ECS

のプラットフォームブランチをベースにした環境のデプロイワークフロー全体を示しています。デプロイのさまざまなフェーズと、各フェーズで Elastic Beanstalk が実行するステップを示します。

前のセクションで説明したワークフローとは異なり、デプロイ設定フェーズでは拡張機能 (`Buildfile` コマンド、`Procfile` コマンド、リバースプロキシ設定) はサポートされていません。

> **メモ**
>
> - この図は、デプロイ中に環境インスタンスで Elastic Beanstalk が実行する一連のステップ全体を表しているわけではありません。この図は、カスタマイズの実行順序とコンテキストを提供するために、説明のために提供します。
> - 簡単にするために、図では、`.platform/hooks/*` フックのサブディレクトリ (アプリケーションデプロイ用) のみを示しており、`.platform/confighooks/*` フックのサブディレクトリ (設定デプロイ用) は示していません。後者のサブディレクトリのフックは、図に示す対応するサブディレクトリのフックとまったく同じステップで実行されます。

デプロイワークフローステップの詳細を次に示します。

a. `EBhooksDir` の `appdeploy/pre` ディレクトリにある実行可能ファイルを実行します。
b. ソースバンドルの `.platform/hooks/prebuild` ディレクトリにある実行可能ファイルを実行します (設定デプロイ用の `.platform/confighooks/prebuild`)。
c. ソースバンドルの `.platform/hooks/predeploy` ディレクトリにある実行可能ファイルを実行します (設定デプロイ用の `.platform/confighooks/predeploy`)。
d. `EBhooksDir` の `appdeploy/enact` ディレクトリにある実行可能ファイルを実行します。
e. `EBhooksDir` の `appdeploy/post` ディレクトリにある実行可能ファイルを実行します。
f. ソースバンドルの `.platform/hooks/postdeploy` ディレクトリにある実行可能ファイルを実行します (設定デプロイ用の `.platform/confighooks/postdeploy`)。

`EBhooksDir` へのリファレンスは、プラットフォームフックディレクトリのパスを表します。ディレクトリパス名を取得するには、次のように、環境インスタンスのコマンドラインで get-config スクリプトツールを実行します。

```
$ /opt/elasticbeanstalk/bin/get-config platformconfig -k EBhooksDir
```

Elastic Beanstalk 環境用のプラットフォームスクリプトツール

このトピックでは、Amazon Linux プラットフォームを使用する環境向けに AWS Elastic Beanstalk が提供するツールについて説明します。これらのツールは Elastic Beanstalk 環境の Amazon EC2 インスタンスにあります。

get-config

ツールを使用して`get-config`、プレーンテキストの環境変数値やその他のプラットフォームやインスタンス情報を取得します。このツールは `/opt/elasticbeanstalk/bin/get-config` にあります。

get-config コマンド

各 `get-config` ツールコマンドは、特定の種類の情報を返します。いずれかのツールのコマンドを実行するには、次の構文を使用します。

```
$ /opt/elasticbeanstalk/bin/get-config command [ options ]
```

次の例では、`environment` コマンドを実行します。

```
$ /opt/elasticbeanstalk/bin/get-config environment -k PORT
```

選択したコマンドとオプションに応じて、ツールはキーバリューのペアを持つオブジェクト (JSON または YAML)、または単一の値を返します。

SSH を使用して Elastic Beanstalk 環境内のEC2 インスタンスに connect することで `get-config` をテストできます。

> **Note**
>
> `get-config` テストを実行する場合、一部のコマンドでは、基礎となる情報にアクセスするために root ユーザー権限が必要になる場合があります。アクセス許可エラーが表示された場合は、`sudo` でコマンドを再度実行します。
> 環境にデプロイするスクリプトでツールを使用するときは、`sudo` を追加する必要はありません。Elastic Beanstalk は、すべてのスクリプトを root ユーザーとして実行します。

以下のセクションでは、各ツールのコマンドについて説明します。

optionsettings - 設定オプション

`get-config optionsettings` コマンドは、環境に設定され、環境インスタンス上のプラットフォームに使用される設定オプションをリストにしたオブジェクトを返します。これらは、名前空間別に組織されています。

```
$ /opt/elasticbeanstalk/bin/get-config optionsettings
{"aws:elasticbeanstalk:application:environment":
{"JDBC_CONNECTION_STRING":""},"aws:elasticbeanstalk:container:tomcat:jvmoptions":{"JVM
 Options":"","Xms":"256m","Xmx":"256m"},"aws:elasticbeanstalk:environment:proxy":
{"ProxyServer":"nginx","StaticFiles":
[""]},"aws:elasticbeanstalk:healthreporting:system":
{"SystemType":"enhanced"},"aws:elasticbeanstalk:hostmanager":
{"LogPublicationControl":"false"}}
```

特定の設定オプションを返すには、`--namespace` (`-n`) オプションを使用して名前空間を指定し、`--option-name` (`-o`) オプションを使用してオプション名を指定します。

```
$ /opt/elasticbeanstalk/bin/get-config optionsettings -
n aws:elasticbeanstalk:container:php:phpini -o memory_limit
256M
```

environment - 環境プロパティ

`get-config environment` コマンドは、ユーザーが設定し、Elastic Beanstalk が提供する環境プロパティのリストを含むオブジェクトを返します。ユーザー設定のプロパティは、コンソールでプレーンテキストとして、または設定オプションの名前空間 を使用して定義されますaws:elasticbeanstalk:application:environment。

```
$ /opt/elasticbeanstalk/bin/get-config environment
{"JDBC_CONNECTION_STRING":"","RDS_PORT":"3306","RDS_HOSTNAME":"anj9aw1b0tbj6b.cijbpanmxz5u.us-
west-2.rds.amazonaws.com","RDS_USERNAME":"testusername","RDS_DB_NAME":"ebdb","RDS_PASSWORD":"te
```

例えば、Elastic Beanstalk は統合された Amazon RDS DB インスタンス (`RDS_HOSTNAME` など) に connect する環境プロパティを提供します。これらの RDS 接続プロパティは、の出力に表示されます。`get-config environment`。ただし、出力には表示されません。`get-config optionsettings`。これは、設定オプションで設定されていないためです。

特定の環境プロパティを返すには、`--key` (`-k`) オプションを使用してプロパティキーを指定します。

```
$ /opt/elasticbeanstalk/bin/get-config environment -k TESTPROPERTY
testvalue
```

> **Note**
>
> このget-configツールは、シークレットを保存する環境変数を取得できません。シークレットストアまたはパラメータストアからプログラムで値を取得する方法の詳細については、Secrets Manager の使用「」または「」を参照してくださいSystems Manager パラメータストアの使用。

container - インスタンス上の構成値

get-config container コマンドは、環境インスタンスのプラットフォームと環境設定値をリストにしたオブジェクトを返します。

次の例は、Amazon Linux 2 Tomcat の環境でのコマンドへの出力を示しています。

```
$ /opt/elasticbeanstalk/bin/get-config container
{"common_log_list":["/var/log/eb-engine.log","/var/log/eb-
hooks.log"],"default_log_list":["/var/log/nginx/access.log","/var/log/nginx/
error.log"],"environment_name":"myenv-1da84946","instance_port":"80","log_group_name_prefix":"/
aws/elasticbeanstalk","proxy_server":"nginx","static_files":
[""],"xray_enabled":"false"}
```

特定のキーの値を返すには、--key (-k) オプションを使用してキーを指定します。

```
$ /opt/elasticbeanstalk/bin/get-config container -k environment_name
myenv-1da84946
```

addons - アドオンの構成値

get-config addons コマンドは、環境アドオンの設定情報を含むオブジェクトを返します。これを使用して、環境に関連付けられた Amazon RDS データベースの設定を回復します。

```
$ /opt/elasticbeanstalk/bin/get-config addons
{"rds":{"Description":"RDS Environment variables","env":
{"RDS_DB_NAME":"ebdb","RDS_HOSTNAME":"ea13k2wimu1dh8i.c18mnpu5rwvg.us-
east-2.rds.amazonaws.com","RDS_PASSWORD":"password","RDS_PORT":"3306","RDS_USERNAME":"user"}}}
```

結果を制限するには、2 つの方法があります。特定のアドオンの値を取得するには、`--add-on (-a)` オプションを使用してアドオン名を指定します。

```
$ /opt/elasticbeanstalk/bin/get-config addons -a rds
{"Description":"RDS Environment variables","env":
{"RDS_DB_NAME":"ebdb","RDS_HOSTNAME":"ea13k2wimu1dh8i.c18mnpu5rwvg.us-
east-2.rds.amazonaws.com","RDS_PASSWORD":"password","RDS_PORT":"3306","RDS_USERNAME":"user"}}
```

アドオン内の特定のキーの値を返すには、`--key (-k)` オプションを追加してキーを指定します。

```
$ /opt/elasticbeanstalk/bin/get-config addons -a rds -k RDS_DB_NAME
ebdb
```

platformconfig - 定数の構成値

`get-config platformconfig` コマンドは、プラットフォームのバージョンに一定のプラットフォーム設定情報を含むオブジェクトを返します。同じプラットフォームのバージョンを実行しているすべての環境で、出力は変わりません。コマンドの出力オブジェクトには、次の 2 つの埋め込んであるオブジェクトがあります。

- `GeneralConfig` - すべての Amazon Linux 2 および Amazon Linux 2023 プラットフォームブランチの最新バージョンで一定の情報が含まれています。
- `PlatformSpecificConfig` - プラットフォームのバージョンに対して一定で、それに固有の情報が含まれます。

次の例は、Tomcat 8.5 running Corretto 11 プラットフォームブランチを使用する環境でのコマンドの出力を示しています。

```
$ /opt/elasticbeanstalk/bin/get-config platformconfig
{"GeneralConfig":{"AppUser":"webapp","AppDeployDir":"/var/app/
current/","AppStagingDir":"/var/app/
staging/","ProxyServer":"nginx","DefaultInstancePort":"80"},"PlatformSpecificConfig":
{"ApplicationPort":"8080","JavaVersion":"11","TomcatVersion":"8.5"}}
```

特定のキーの値を返すには、`--key (-k)` オプションを使用してキーを指定します。これらのキーは、2 つの埋め込みオブジェクト間で一意です。キーを含むオブジェクトを指定する必要はありません。

```
$ /opt/elasticbeanstalk/bin/get-config platformconfig -k AppStagingDir
```

```
/var/app/staging/
```

get-config 出力オプション

出力オブジェクトの形式を指定するには、`--output` オプションを使用します。有効な値は、`JSON` (デフォルト) と `YAML` です。これはグローバルオプションです。コマンド名の前に指定する必要があります。

次の例は、設定オプション値を YAML 形式で返すものです。

```
$ /opt/elasticbeanstalk/bin/get-config --output YAML optionsettings
aws:elasticbeanstalk:application:environment:
  JDBC_CONNECTION_STRING: ""
aws:elasticbeanstalk:container:tomcat:jvmoptions:
  JVM Options: ""
  Xms: 256m
  Xmx: 256m
aws:elasticbeanstalk:environment:proxy:
  ProxyServer: nginx
  StaticFiles:
        - ""
aws:elasticbeanstalk:healthreporting:system:
  SystemType: enhanced
aws:elasticbeanstalk:hostmanager:
  LogPublicationControl: "false"
```

pkg-repo

> **Note**
>
> `pkg-repo` ツールは、Amazon Linux 2023 プラットフォームに基づく環境では使用できません。ただし、パッケージとオペレーティングシステムの更新を AL2023 インスタンスに手動で適用することはできます。詳細については、「Amazon Linux 2023 ユーザーガイド」の「パッケージとオペレーティングシステムの更新の管理」を参照してください

緊急の状況によっては、必要な Elastic Beanstalk プラットフォームバージョンでまだリリースされていない Amazon Linux 2 セキュリティパッチで Amazon EC2 インスタンスを更新する必要がある場合があります。デフォルトでは、Elastic Beanstalk 環境で手動更新を実行することはできません。これは、プラットフォームのバージョンが Amazon Linux 2 リポジトリの特定のバージョンにロック

されているためです。このロックにより、インスタンスがサポートされ、一貫性のあるソフトウェアバージョンが実行されることが保証されます。緊急の場合、`pkg-repo`ツールを使用すると、新しい Elastic Beanstalk プラットフォームバージョンでリリースされる前に環境にインストールする必要がある場合は、Amazon Linux 2 で yum パッケージを手動で更新する回避策を使用できます。

-`pkg-repo`Amazon Linux 2 プラットフォーム上のツールは、`yum`パッケージリポジトリ。その後、を手動で実行できますyum updateセキュリティパッチの場合。逆に、ツールを使用して yum パッケージのリポジトリをロックして、さらなる更新を防ぐことで、更新をフォローできます。-`pkg-repo`このツールはにあります`/opt/elasticbeanstalk/bin/pkg-repo`Elastic Beanstalk 環境にあるすべての EC2 インスタンスのディレクトリ。

を使用した変更`pkg-repo`ツールは、ツールが使用されている EC2 インスタンスでのみ作成されます。他のインスタンスに影響したり、環境への今後の更新を妨げたりすることはありません。このトピックで後述する例では、を呼び出して、すべてのインスタンスに変更を適用する方法を説明します。`pkg-repo`スクリプトおよび設定ファイルからのコマンド

> **Warning**
>
> このツールは推奨されませんユーザー。ロック解除されたプラットフォームバージョンに適用される手動による変更は、帯域外と見なされます。このオプションは、次のリスクを受け入れる可能性のある緊急の状況にあるユーザーに対してのみ実行できます。
>
> - プラットフォームのバージョンは、環境内のすべてのインスタンスで一貫性が保証されるわけではありません。
> - を使用して変更された環境`pkg-repo`ツールが正しく機能することは保証されません。Elastic Beanstalk がサポートするプラットフォームではテストおよび検証が行われていません。
>
> テストとバックアウト計画を含むベストプラクティスを適用することを強くお勧めします。ベストプラクティスの促進に役立つ,Elastic Beanstalk コンソールと EB CLI を使用すると、環境のクローンを作成し、環境 URL をスワップできます。これらの操作の詳細については、このガイドの「環境の管理」の章の「ブルー/グリーンデプロイ」を参照してください。

yum リポジトリ構成ファイルを手動で編集する場合は、`pkg-repo`ツールはまずです。-`pkg-repo`yum リポジトリ設定ファイルを手動で編集した Amazon Linux 2 環境では、ツールが意図したとおりに動作しない場合があります。これは、ツールが構成の変更を認識しない可能性があるためです。

Amazon Linux パッケージリポジトリについては、「Amazon EC2 ユーザーガイド」の「パッケージリポジトリ」のトピックを参照してください。

pkg-repo コマンド

pkg-repoツールのコマンドを実行するには、次の構文を使用します。

```
$ /opt/elasticbeanstalk/bin/pkg-repo command [options]
```

pkg-repo コマンドを以下に示します：

- lock— をロックしますyumリポジトリを特定のバージョンにパッケージ化する
- unlock— ロックを解除しますyum特定のバージョンのリポジトリをパッケージ化する
- status— すべてのリストを表示します。yumパッケージリポジトリとその現在のロックステータス
- help— 1 つのコマンドに関する一般的なヘルプまたはヘルプを表示します

オプションは、以下のようにコマンドに適用されます。

- lock,unlockそしてstatus — オプション:-h,--help、またはなし (既定)。
- help— オプション:lock,unlock,status、またはなし (既定)。

次の例では、unlock コマンドを実行します。

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo unlock
Amazon Linux 2 core package repo successfully unlocked
Amazon Linux 2 extras package repo successfully unlocked
```

次の例では、lock コマンドを実行します。

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo lock
Amazon Linux 2 core package repo successfully locked
Amazon Linux 2 extras package repo successfully locked
```

次の例では、status コマンドを実行します。

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo status
Amazon Linux 2 core package repo is currently UNLOCKED
```

```
Amazon Linux 2 extras package repo is currently UNLOCKED
```

次の例では、helpコマンドのlockコマンド。

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo help lock
```

次の例では、helpコマンドのpkg-repoツール。

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo help
```

SSH を使用して Elastic Beanstalk 環境内のインスタンスに接続することで pkg-repo をテストできます。SSH オプションの 1 つに EB CLI がありますeb sshコマンド。

> **Note**
>
> -pkg-repoツールを実行するには root ユーザー権限が必要です。アクセス許可エラーが表示された場合は、sudo でコマンドを再度実行します。
> 環境にデプロイするスクリプトまたは設定ファイルでツールを使用するときは、sudo を追加する必要はありません。Elastic Beanstalk は、すべてのスクリプトを root ユーザーとして実行します。

pkg-repo の例

前のセクションでは、Elastic Beanstalk 環境の個々の EC2 インスタンスでテストするためのコマンドラインの例を示します。このアプローチはテストに役立ちます。ただし、一度に更新されるインスタンスは 1 つだけなので、環境内のすべてのインスタンスに変更を適用するのは現実的ではありません。

より実用的なアプローチはプラットフォームフックスクリプトまたは.ebextensions一貫した方法ですべてのインスタンスに変更を適用するための設定ファイル。

次の例では、を呼び出します。pkg-repoの設定ファイルから.ebextensionsfolder Elastic Beanstalk は、update_package.configアプリケーションソースバンドルをデプロイするときにファイルを指定します。

```
.ebextensions
### update_package.config
```

docker パッケージが最新バージョンを受け取るには、この設定で、yum update コマンドの docker パッケージを指定します。

```
### update_package.config ###

commands:
  update_package:
    command: |
      /opt/elasticbeanstalk/bin/pkg-repo unlock
      yum update docker -y
      /opt/elasticbeanstalk/bin/pkg-repo lock
      yum clean all -y
      rm -rf /var/cache/yum
```

この設定では、yum updateコマンド。その結果、利用可能なすべての更新が適用されます。

```
### update_package.config ###

commands:
  update_package:
    command: |
      /opt/elasticbeanstalk/bin/pkg-repo unlock
      yum update -y
      /opt/elasticbeanstalk/bin/pkg-repo lock
      yum clean all -y
      rm -rf /var/cache/yum
```

次の例では、を呼び出します。`pkg-repo`bashスクリプトからプラットフォームフック。Elastic Beanstalk`update_package.sh`にあるスクリプトファイル`prebuild`サブディレクトリ。

```
.platform
### hooks
    ### prebuild
        ### update_package.sh
```

docker パッケージが最新バージョンを受け取るには、このスクリプトで yum update コマンドの docker パッケージを指定します。パッケージ名を省略すると、すべての利用可能な更新が適用されます。前の設定ファイルの例では、この方法を示しています。

```
### update_package.sh ###
```

```
#!/bin/bash

/opt/elasticbeanstalk/bin/pkg-repo unlock
yum update docker -y
/opt/elasticbeanstalk/bin/pkg-repo lock
yum clean all -y
rm -rf /var/cache/yum
```

download-source-bundle (Amazon Linux AMI のみ)

Amazon Linux AMI プラットフォームブランチ (Amazon Linux 2 より前) では、Elastic Beanstalk は追加ツールを提供していて、それはdownload-source-bundleです。このツールを使用して、プラットフォームのデプロイ中にアプリケーションの出典コードをダウンロードします。このツールは `/opt/elasticbeanstalk/bin/download-source-bundle` にあります。

サンプルスクリプト `00-unzip.sh` は、環境インスタンスの `appdeploy/pre` フォルダにあります。これは `download-source-bundle` を使用して、デプロイ中にアプリケーションの出典コードを `/opt/elasticbeanstalk/deploy/appsource` フォルダにダウンロードする方法を示しています。

Elastic Beanstalk Linux プラットフォームの拡張

このトピックでは、独自のコマンド、スクリプト、ソフトウェア、設定を使用して Linux プラットフォームを拡張する方法について説明します。デフォルトのプロキシサーバーと設定を変更するには、プラットフォームを拡張する必要がある場合があります。または、プラットフォームがアプリケーションをビルドまたは起動する方法をカスタマイズする必要がある場合があります。

トピック

- ビルドファイルと Procfile
- プラットフォームフック
- 設定ファイル
- リバースプロキシの設定
- 拡張機能を使用したアプリケーションの例

ビルドファイルと Procfile

プラットフォームによっては、アプリケーションの構築または準備方法をカスタマイズしたり、アプリケーションを実行するプロセスを指定することができます。各プラットフォームのトピックでは、

プラットフォームによりサポートされている場合は特に Buildfile や Procfile について言及されています。特定のプラットフォームについては、「[Platforms] (プラットフォーム)」を参照してください。

このページで説明されているとおり、サポートされているすべてのプラットフォームでは、構文とセマンティクスは同じです。個々のプラットフォームのトピックでは、それぞれの言語でアプリケーションを構築および実行するために、ファイルの具体的な使用方法について説明しています。

[Buildfile]

アプリケーションのカスタムビルドおよび設定コマンドを指定するには、アプリケーションソースのルートディレクトリに `Buildfile` という名前のファイルを配置します。ファイル名では、大文字と小文字が区別されます。`Buildfile` には、次の構文を使用します。

```
<process_name>: <command>
```

`Buildfile` 内のコマンドは、正規表現 `^[A-Za-z0-9_-]+:\s*[^\s].*$` に一致する必要があります。

Elastic Beanstalk は、`Buildfile` で実行されるアプリケーションをモニタリングしません。`Buildfile` は、短期間実行されてタスクが完了したら終了されるコマンドに使用します。長期間継続的に実行される必要のあるアプリケーションプロセスには、Procfile を使用します。

`Buildfile` 内のすべてのパスは、出典バンドルのルートと関連します。次の `Buildfile` の例では、`build.sh` がシェルスクリプトとして、ソースバンドルのルートに配置されています。

Example [Buildfile]

```
make: ./build.sh
```

カスタムビルドステップを提供する場合は、`Buildfile` ではなく、最も単純なコマンド以外の `predeploy` プラットフォームフックを使用することをお勧めします。プラットフォームフックは、より豊富なスクリプトとより良いエラー処理を可能にします。プラットフォームフックについては、次のセクションで説明します。

[Procfile]

アプリケーションを起動して実行するためのカスタムコマンドを指定するには、アプリケーションソースのルートディレクトリに `Procfile` という名前のファイルを配置します。ファイル名では、

大文字と小文字が区別されます。Procfile には、次の構文を使用します。1 つまたは複数のコマンドを指定できます。

```
<process_name1>: <command1>
<process_name2>: <command2>
...
```

Procfile 内の各行は、正規表現 ^[A-Za-z0-9_-]+:\s*[^\s].*$ に一致する必要があります。

長時間継続的に実行されるアプリケーションプロセスには、Procfile を使用します。Elastic Beanstalk では、Procfile のプロセスは継続的に実行される必要があります。Elastic Beanstalk はこれらのプロセスをモニタリングし、終了されたプロセスをすべて再開します。短期間実行されるプロセスには、Buildfile コマンドを使用します。

Procfile 内のすべてのパスは、出典バンドルのルートと関連します。次の例では、Procfile は 3 つのプロセスを定義します。最初のプロセスは、この例では web と呼ばれますが、メインウェブアプリケーションです。

Example [Procfile]

```
web: bin/myserver
cache: bin/mycache
foo: bin/fooapp
```

Elastic Beanstalk は、ポート 5000 のメインウェブアプリケーションにリクエストを転送するようにプロキシサーバーを設定し、このポート番号を設定できます。Procfile の一般的な使用方法は、このポート番号をコマンド引数としてアプリケーションに渡すことです。プロキシ設定の詳細については、「リバースプロキシの設定」を参照してください。

Elastic Beanstalk は、Procfile プロセスから標準出力ストリームとエラーストリームをログファイルにキャプチャします。Elastic Beanstalk は、プロセスの後にログファイルに名前を付け、/var/log に保存します。たとえば、前述の例では web プロセスが web-1.log および web-1.error.log についてそれぞれ stdout および stderr という名前のログを生成します。

プラットフォームフック

プラットフォームフックは、環境のプラットフォームを拡張するために特別に設計されています。これらは、アプリケーションのソースコードの一部としてデプロイするカスタムスクリプトおよび他の

実行可能ファイルで、Elastic Beanstalk によって、さまざまなインスタンスプロビジョニング段階で実行されます。

> **Note**
>
> プラットフォームフックは、Amazon Linux AMI プラットフォームのバージョン (Amazon Linux 2 より前) ではサポートされていません。

アプリケーションのデプロイプラットフォームフック

アプリケーションのデプロイは、デプロイ用の新しいソースバンドルを提供する場合、またはすべての環境インスタンスの終了および再作成が必要な設定の変更を行う場合に発生します。

アプリケーションのデプロイ中に実行されるプラットフォームフックを提供するには、ソースバンドルの `.platform/hooks` ディレクトリの下にある以下のサブディレクトリのいずれかにファイルを配置します。

- `prebuild` - ファイルは、Elastic Beanstalk プラットフォームエンジンがアプリケーションソースバンドルをダウンロードして抽出した後、アプリケーションとウェブサーバーをセットアップおよび設定する前に実行されます。

  `prebuild` ファイルは、設定ファイルの commandsセクションにあるコマンドを実行した後、および `Buildfile` コマンドを実行する前に実行されます。
- `predeploy` - ここでのファイルは、Elastic Beanstalk アプリケーションとウェブサーバーをセットアップして設定した後、最終的なランタイムの場所にデプロイする前に実行されます。

  `predeploy` ファイルは、設定ファイルの container_commandsセクションにあるコマンドを実行した後、および `Procfile` コマンドを実行する前に実行されます。
- `postdeploy` - ここでのファイルは、Elastic Beanstalk プラットフォームエンジンがアプリケーションとプロキシサーバーをデプロイした後に実行されます。

  これは、最後のデプロイワークフローのステップです。

設定デプロイプラットフォームフック

設定デプロイは、環境インスタンスを再作成せずに更新だけする設定変更を行った場合に発生します。次のオプションの変更は、設定を更新します。

- 環境プロパティとプラットフォーム固有の設定
- 静的ファイル
- AWS X-Ray デーモン
- ログストレージおよびストリーミング
- アプリケーションポート (詳細については、「リバースプロキシの設定」を参照してください)

設定のデプロイ中に実行されるフックを提供するには、ソースバンドルの `.platform/confighooks` ディレクトリの下にフックを配置します。アプリケーションのデプロイフックと同じ 3 つのサブディレクトリが適用されます。

プラットフォームフックの詳細

フックファイルは、バイナリファイル、またはインタプリタパスを含む `#!` 行で始まるスクリプトファイル (`#!/bin/bash` など) です。すべてのファイルには、実行アクセス許可が必要です。フックファイルの実行アクセス許可を設定するために `chmod +x` を使用します。2022 年 4 月 29 日以降にリリースされたすべての Amazon Linux 2023 および Amazon Linux 2 ベースのプラットフォームバージョンでは、Elastic Beanstalk はすべてのプラットフォームフックスクリプトに対して実行アクセス権限を自動的に付与します。この場合、実行アクセス権限を手動で付与する必要はありません。これらのプラットフォームのバージョンのリストについては、「AWS Elastic Beanstalk リリースノートガイド」で 2022 年 4 月 29 日 の Linux リリースノートを参照してください。

Elastic Beanstalk は、ファイル名の辞書順でこれらのディレクトリの各ファイルを実行します。すべてのファイルが `root` ユーザーとして実行されます。プラットフォームフックの現在の作業ディレクトリ (cwd) は、アプリケーションのルートディレクトリです。`prebuild` および `predeploy` ファイルの場合はアプリケーションのステージングディレクトリで、`postdeploy` ファイルの場合は現在のアプリケーションディレクトリです。いずれかのファイルが失敗した場合 (ゼロ以外の終了コードで終了した場合) 、デプロイは中止され、失敗します。

プラットフォームフックテキストスクリプトに Windows キャリッジリターン/ラインフィード (CRLF) 改行文字が含まれていると、失敗することがあります。ファイルを Windows ホストに保存してから Linux サーバーに転送した場合、ファイルには Windows CRLF の改行が含まれていることがあります。2022 年 12 月 29 日以降にリリースされたプラットフォームでは、Elastic Beanstalk でプラットフォームフックテキストファイル内の Windows CRLF 文字を Linux ラインフィード (LF) 改行文字に自動的に変換します。この日付より前にリリースされた Amazon Linux 2 プラットフォーム上でアプリケーションを実行する場合は、Windows CRLF 文字を Linux LF 文字に変換する必要があります。これを実現する方法の 1 つは、Linux ホストでスクリプトファイルを作成して保存することです。これらの文字を変換するツールはインターネットでも入手できます。

フックファイルは、アプリケーションオプションで定義したすべての環境プロパティ、およびシステム環境変数 HOME、PATH、および PORT にアクセスできます。

環境変数やその他の設定オプションの値をプラットフォームフックスクリプトに取り込むには、Elastic Beanstalk が環境インスタンスに提供する get-config ユーティリティを使用できます。詳細については、「the section called “プラットフォームスクリプトツール”」を参照してください。

設定ファイル

アプリケーションのソースコードの .ebextensions ディレクトリに設定ファイルを追加して、Elastic Beanstalk 環境のさまざまな側面を設定できます。とりわけ、設定ファイルを使用すると、環境のインスタンス上のソフトウェアやその他のファイルをカスタマイズしたり、インスタンスで初期化コマンドを実行できます。詳細については、「the section called “[Linux サーバー]”」を参照してください。

設定ファイルを使用して、設定オプションを設定することもできます。オプションの多くはプラットフォームの動作を制御し、これらのオプションの一部はプラットフォーム固有のものです。

Amazon Linux 2 および Amazon Linux 2023 に基づくプラットフォームの場合、インスタンスをプロビジョニング中の環境インスタンスでカスタムコードを設定および実行するには、Buildfile、Procfile、およびプラットフォームフックをご使用になることをお勧めします。これらのメカニズムについては、このページの前のセクションで説明しています。.ebextensions 設定ファイルでコマンドやコンテナコマンドを使用することはできますが、操作は簡単ではありません。たとえば、YAML ファイル内にコマンドスクリプトを記述することは、構文の観点から難しい場合があります。 AWS CloudFormation リソースへの参照が必要なスクリプトには、引き続き.ebextensions設定ファイルを使用する必要があります。

リバースプロキシの設定

すべての Amazon Linux 2 および Amazon Linux 2023 プラットフォームのバージョンは、デフォルトのリバースプロキシサーバーとして nginx を使用します。Tomcat、Node.js、PHP、Python のプラットフォームも、代替として Apache HTTPD をサポートします。これらのプラットフォームで Apache を選択するには、aws:elasticbeanstalk:environment:proxy 名前空間の ProxyServer オプションを apache に設定します。このセクションの説明の通り、すべてのプラットフォームでプロキシサーバーの設定が一貫して有効にされています。

> **Note**
>
> Amazon Linux AMI プラットフォームのバージョン (Amazon Linux 2 より前) では、異なるプロキシサーバー設定にすることが必要になる場合があります。これらのレガシーの詳細は、このガイドのそれぞれのプラットフォームのトピックを参照してください。

Elastic Beanstalk は、環境のルート URL 上のメインウェブアプリケーションにウェブトラフィックを転送するように、環境のインスタンスでプロキシサーバーを設定します (例: `http://my-env.elasticbeanstalk.com`)。

デフォルトでは、Elastic Beanstalk はポート 80 に届くリクエストをポート 5000 のメインウェブアプリケーションに送信するようにプロキシを設定します。次の例に示すように、設定ファイルで aws:elasticbeanstalk: application: environment 名前空間を使用して `PORT` 環境プロパティを設定することによって、このポート番号を設定できます。

```
option_settings:
  - namespace:  aws:elasticbeanstalk:application:environment
    option_name:  PORT
    value:  <main_port_number>
```

使用しているアプリケーションの環境変数の設定の詳細については、「the section called “オプション設定”」を参照してください。

アプリケーションは、プロキシで設定されているポートをリッスンする必要があります。`PORT` 環境プロパティを使用してデフォルトポートを変更した場合、コードは `PORT` 環境変数の値を読み取ることによってポートにアクセスできます。たとえば、Go の `os.Getenv("PORT")` または Java の `System.getenv("PORT")` を呼び出します。複数のアプリケーションプロセスにトラフィックを送信するようにプロキシを設定する場合、複数の環境プロパティを設定し、それらの値をプロキシ設定とアプリケーションコードの両方で使用できます。もう 1 つのオプションは、`Procfile` でコマンド引数としてポート値をプロセスに渡すことです。詳細については、「ビルドファイルと Procfile」を参照してください。

nginx の設定

Elastic Beanstalk は、デフォルトのリバースプロキシとして nginx を使用し、アプリケーションを Elastic Load Balancing ロードバランサーにマッピングします。Elastic Beanstalk は、拡張または独自の設定で完全に上書きできるデフォルトの nginx 設定を提供します。

> **Note**
>
> nginx `.conf` 設定ファイルを追加または編集するときは、必ず UTF-8 としてエンコードしてください。

Elastic Beanstalk のデフォルトの nginx 設定を拡張するには、アプリケーションソースバンドルの `.platform/nginx/conf.d/` というフォルダに `.conf` 設定ファイルを追加します。Elastic Beanstalk の nginx 設定では、このフォルダに `.conf` ファイルが自動的に含められます。

```
~/workspace/my-app/
|-- .platform
|   `-- nginx
|       `-- conf.d
|           `-- myconf.conf
`-- other source files
```

Elastic Beanstalk のデフォルトの nginx 設定を完全に上書きするには、ソースバンドルの `.platform/nginx/nginx.conf` に設定を含めます。

```
~/workspace/my-app/
|-- .platform
|   `-- nginx
|       `-- nginx.conf
`-- other source files
```

Elastic Beanstalk の nginx 設定を上書きするには、`nginx.conf` に以下の行を追加することにより、Elastic Beanstalk でのヘルスレポートとモニタリングの強化、自動アプリケーションマッピング、および静的ファイルに関して Elastic Beanstalk の設定を適用します。

```
include conf.d/elasticbeanstalk/*.conf;
```

Apache HTTPD の設定

Tomcat、Node.js、PHP、Python のプラットフォームでは、nginx の代わりに Apache HTTPD プロキシサーバーを選択できます。これはデフォルトではありません。次の例では、Apache HTTPD が使用されるように Elastic Beanstalk を設定しています。

Example .ebextensions/httpd-proxy.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
```

追加の設定ファイルを使用して、Elastic Beanstalk のデフォルト Apache 設定を拡張できます。または、Elastic Beanstalk のデフォルトの Apache 設定を完全に上書きすることもできます。

Elastic Beanstalk のデフォルトの Apache 設定を拡張するには、アプリケーションソースバンドルの `.conf` というフォルダに `.platform/httpd/conf.d` 設定ファイルを追加します。Elastic Beanstalk の Apache 設定では、このフォルダに `.conf` ファイルが自動的に含められます。

```
~/workspace/my-app/
|-- .ebextensions
|   -- httpd-proxy.config
|-- .platform
|   -- httpd
|      -- conf.d
|         -- port5000.conf
|         -- ssl.conf
-- index.jsp
```

たとえば、次の Apache 2.4 設定では、ポート 5000 にリスナーを追加します。

Example .platform/httpd/conf.d/port5000.conf

```
listen 5000
<VirtualHost *:5000>
  <Proxy *>
    Require all granted
  </Proxy>
  ProxyPass / http://localhost:8080/ retry=0
  ProxyPassReverse / http://localhost:8080/
  ProxyPreserveHost on

  ErrorLog /var/log/httpd/elasticbeanstalk-error_log
</VirtualHost>
```

Elastic Beanstalk のデフォルトの Apache 設定を完全に上書きするには、ソースバンドルの `.platform/httpd/conf/httpd.conf` に設定を含めます。

```
~/workspace/my-app/
|-- .ebextensions
|   -- httpd-proxy.config
|-- .platform
|   `-- httpd
|       `-- conf
|           `-- httpd.conf
`-- index.jsp
```

> **Note**
>
> Elastic Beanstalk の Apache 設定を上書きするには、`httpd.conf` に以下の行を追加することにより、Elastic Beanstalk でのヘルスレポートとモニタリングの強化、自動アプリケーションマッピング、および静的ファイルに関して Elastic Beanstalk の設定を適用します。
>
> ```
> IncludeOptional conf.d/elasticbeanstalk/*.conf
> ```

拡張機能を使用したアプリケーションの例

以下の例は、Elastic Beanstalk Amazon Linux 2 および Amazon Linux 2023 プラットフォームでサポートされるいくつかの拡張機能 (`Procfile`、`.ebextensions` 設定ファイル、カスタムフック、およびプロキシ設定ファイル) を備えたアプリケーションソースバンドルを示しています。

```
~/my-app/
|-- web.jar
|-- Procfile
|-- readme.md
|-- .ebextensions/
|   |-- options.config        # Option settings
|   `-- cloudwatch.config     # Other .ebextensions sections, for example files and
 container commands
`-- .platform/
    |-- nginx/                # Proxy configuration
    |   |-- nginx.conf
    |   `-- conf.d/
    |       `-- custom.conf
    |-- hooks/                # Application deployment hooks
    |   |-- prebuild/
    |   |   |-- 01_set_secrets.sh
```

```
    |   |   `-- 12_update_permissions.sh
    |   |-- predeploy/
    |   |   `-- 01_some_service_stop.sh
    |   `-- postdeploy/
    |       |-- 01_set_tmp_file_permissions.sh
    |       |-- 50_run_something_after_app_deployment.sh
    |       `-- 99_some_service_start.sh
    `-- confighooks/          # Configuration deployment hooks
        |-- prebuild/
        |   `-- 01_set_secrets.sh
        |-- predeploy/
        |   `-- 01_some_service_stop.sh
        `-- postdeploy/
            |-- 01_run_something_after_config_deployment.sh
            `-- 99_some_service_start.sh
```

> **Note**
>
> これらの拡張機能のいくつかは、Amazon Linux AMI プラットフォームのバージョン (Amazon Linux 2 より前) ではサポートされていません。

Elastic Beanstalk を使用した .NET Windows アプリケーションのデプロイ

> **デ AWS ベロッパーセンターの .NET を確認する**
>
> .Net デベロッパーセンターにアクセスしたことがありますか? これは、.NET on のすべてのモノのワンストップショップです AWS。
> 詳細については、 AWS デベロッパーセンターの .NET を参照してください。

この章では、ASP.NET および .NET Core Windows ウェブアプリケーションを設定してデプロイする手順について説明します AWS Elastic Beanstalk。Elastic Beanstalk を使用すると、Amazon Web Services を使用して .NET (Windows) ウェブアプリケーションのデプロイ、管理、スケーリングを簡単に行うことができます。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) または Elastic Beanstalk コンソールを使用すると、わずか数分でアプリケーションをデプロイできます。Elastic Beanstalk アプリケーションをデプロイした後も、EB CLI を使用してアプリケーションと環境を管理することも、Elastic Beanstalk コンソール AWS CLIまたは APIs を使用することもできます。

この章には、以下のチュートリアルが含まれます。

- .NET Core on Windows の QuickStart – EB CLI を使用して Hello World .NET Core Windows アプリケーションを作成およびデプロイするステップバイステップの手順。
- ASP.NET の QuickStart – AWS Toolkit for Visual Studioを使用して Hello World ASP.NET アプリケーションを作成およびデプロイするステップバイステップの手順。

Windows .NET Core アプリケーションの開発に関するヘルプが必要な場合は、次のようにいくつかの参照先があります。

- .NET 開発フォーラム – 質問を投稿し、フィードバックを取得します。
- .NET デベロッパーセンター – サンプルコード、ドキュメント、ツール、追加リソースを 1 か所で入手できる場所です。
- AWS SDK for .NET ドキュメント — SDK のセットアップとコードサンプルの実行、SDK の機能、および SDK の API オペレーションに関する詳細情報について説明します。

> **Note**
>
> このプラットフォームはワーカー環境をサポートしていません。詳細については、Elastic Beanstalk ワーカー環境 を参照してください

トピック

- QuickStart: Elastic Beanstalk に .NET Core on Windows アプリケーションをデプロイする
- QuickStart: Elastic Beanstalk に ASP.NET アプリケーションをデプロイする
- .NET 開発環境のセットアップ
- Elastic Beanstalk .NET Windows プラットフォームの使用
- .NET アプリケーション環境に Amazon RDS DB インスタンスを追加
- AWS Toolkit for Visual Studio
- オンプレミス .NET アプリケーションの Elastic Beanstalk への移行
- Elastic Beanstalk での Windows Server の廃止されたコンポーネントに関する推奨事項

QuickStart: Elastic Beanstalk に .NET Core on Windows アプリケーションをデプロイする

この QuickStart チュートリアルでは、.NET Core on Windows アプリケーションを作成し、 AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

> **Note**
>
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: .NET Core on Windows アプリケーションを作成する
- ステップ 2: アプリケーションをローカルに実行する

- ステップ 3: EB CLI を使用して .NET Core on Windows アプリケーションをデプロイする
- ステップ 4: Elastic Beanstalk でアプリケーションを実行する
- ステップ 5：クリーンアップ
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

AWS アカウントをすでにお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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、通話呼び出し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するように求められます。

   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 Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 「 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 (>) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
C:\eb-project> this is a command
this is output
```

EB CLI

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

.NET Core on Windows

ローカルマシンに .NET SDK がインストールされていない場合は、.NET ドキュメントウェブサイトの .NET のダウンロードリンクに従ってインストールできます。

次のコマンドを実行して、.NET SDK のインストールを確認します。

```
C:\> dotnet --info
```

ステップ 1: .NET Core on Windows アプリケーションを作成する

プロジェクトディレクトリを作成します。

```
C:\> mkdir eb-dotnetcore
C:\> cd eb-dotnetcore
```

次に、次のコマンドを実行して、サンプルの Hello World RESTful ウェブサービスアプリケーションを作成します。

```
C:\eb-dotnetcore> dotnet new web --name HelloElasticBeanstalk
C:\eb-dotnetcore> cd HelloElasticBeanstalk
```

ステップ 2: アプリケーションをローカルに実行する

アプリケーションをローカルで実行するには、次のコマンドを実行します。

```
C:\eb-dotnetcore\HelloElasticBeasntalk> dotnet run
```

出力は次のテキストのようになります。

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7222
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5228
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\Administrator\eb-dotnetcore\HelloElasticBeanstalk
```

> **Note**
>
> dotnet コマンドは、アプリケーションをローカルで実行するときに、ポートをランダムに選択します。この例では、ポートは 5228 です。アプリケーションを Elastic Beanstalk 環境にデプロイすると、アプリケーションはポート 5000 で実行されます。

ウェブブラウザに URL アドレス `http://localhost:port` を入力します。この特定の例では、コマンドは `http://localhost:5228` です。ウェブブラウザに「Hello World!」と表示されます。

ステップ 3: EB CLI を使用して .NET Core on Windows アプリケーションをデプロイする

次のコマンドを実行して、このアプリケーションの Elastic Beanstalk 環境を作成します。

環境を作成し、.NET Core on Windows アプリケーションをデプロイするには

1. `HelloElasticBeanstalk` ディレクトリで次のコマンドを実行して、アプリケーションを公開して圧縮します。

```
C:\eb-dotnetcore\HelloElasticBeasntalk> dotnet publish -o site
C:\eb-dotnetcore\HelloElasticBeasntalk> cd site
C:\eb-dotnetcore\HelloElasticBeasntalk\site> Compress-Archive -Path * -
DestinationPath ../site.zip
C:\eb-dotnetcore\HelloElasticBeasntalk\site> cd ..
```

2. `aws-windows-deployment-manifest.json` という新しいファイルを `HelloElasticBeanstalk` に次の内容で作成します。

```
{
    "manifestVersion": 1,
    "deployments": {
        "aspNetCoreWeb": [
        {
            "name": "test-dotnet-core",
            "parameters": {
                "appBundle": "site.zip",
                "iisPath": "/",
                "iisWebSite": "Default Web Site"
            }
        }
        ]
    }
}
```

3. eb init コマンドを使用して EB CLI リポジトリを初期化します。

```
C:\eb-dotnetcore\HelloElasticBeasntalk> eb init -p iis dotnet-windows-server-
tutorial --region us-east-2
```

このコマンドは、dotnet-windows-server-tutorial という名前のアプリケーションを作成し、ローカルリポジトリを設定して最新の Windows Server プラットフォームバージョンで環境を作成します。

4. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。Elastic Beanstalk は、アプリケーションの zip ファイルを自動的にビルドし、ポート 5000 で起動します。

```
C:\eb-dotnetcore\HelloElasticBeasntalk> eb create dotnet-windows-server-env
```

Elastic Beanstalk が環境を作成するのに約 5 分かかります。

ステップ 4: Elastic Beanstalk でアプリケーションを実行する

環境を作成するプロセスが完了したら、eb open でウェブサイトを開きます。

```
C:\eb-dotnetcore\HelloElasticBeasntalk> eb open
```

お疲れ様でした。Elastic Beanstalk を使用して .NET Core on Windows アプリケーションをデプロイしました。これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ Window が開きます。

ステップ 5：クリーンアップ

アプリケーションでの作業が終了したら、環境を終了できます。Elastic Beanstalk は、環境に関連付けられているすべての AWS リソースを終了します。

EB CLI を使用して Elastic Beanstalk 環境を終了するには、次のコマンドを実行します。

```
C:\eb-dotnetcore\HelloElasticBeasntalk> eb terminate
```

AWS アプリケーションの リソース

1 つのインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありませ

ん。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えま

す。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

> **その他のチュートリアルを試す**
>
> 異なるアプリケーション例の他のチュートリアルを試したい場合は、「ASP.NET の QuickStart」を参照してください。

1 つか 2 つのサンプルアプリケーションをデプロイし、ローカルで .NET Core on Windows アプリケーションを開発して実行する準備が整ったら、「.NET 開発環境のセットアップ」を参照します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。

QuickStart: Elastic Beanstalk に ASP.NET アプリケーションをデプロイする

この QuickStart チュートリアルでは、ASP.NET アプリケーションを作成し、 AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

> **Note**
>
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにはアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: ASP.NET アプリケーションを作成する
- ステップ 2: アプリケーションをローカルに実行する

- ステップ 3: を使用して ASP.NET アプリケーションをデプロイする AWS Toolkit for Visual Studio
- ステップ 4: Elastic Beanstalk でアプリケーションを実行する
- ステップ 5：クリーンアップ
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

AWS アカウントをすでにお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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、通話呼び出し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するように求められます。

   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 Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

この QuickStart チュートリアルでは、「Hello World」アプリケーションを作成し、Visual Studio とを使用して Elastic Beanstalk 環境にデプロイする方法について説明します AWS Toolkit for Visual Studio。

Visual Studio

Visual Studio をダウンロードしてインストールするには、Visual Studio ダウンロードページの手順に従います。この例では、Visual Studio 2022 を使用しています。Visual Studio のインストール中に、以下の特定の項目を選択します。

- [ワークロード] タブで、[ASP.NET とウェブ開発] を選択します。
- [個別コンポーネント] タブで、[.NET Framework 4.8 開発ツール] と [.NET Framework プロジェクトと項目テンプレート] を選択します。

AWS Toolkit for Visual Studio

ダウンロードしてセットアップするには、「 AWS Toolkit for Visual Studio ユーザーガイド」の「開始方法」トピックの手順 AWS Toolkit for Visual Studio に従います。

ステップ 1: ASP.NET アプリケーションを作成する

次に、Elastic Beanstalk 環境にデプロイするアプリケーションを作成します。ここでは「Hello World」ASP.NET ウェブアプリケーションを作成します。

ASP.NET アプリケーションを作成するには

1. Visual Studio を起動します。[ファイル] メニューで、[新規] を選択し、次に [プロジェクト] を選択します。
2. [新しいプロジェクトの作成] ダイアログボックスが表示されます。[ASP.NET ウェブアプリケーション (.NET Framework)] を選択し、[次へ] を選択します。
3. [新しいプロジェクトの設定] ダイアログで、[プロジェクト名] に `eb-aspnet` と入力します。[フレームワーク] ドロップダウンメニューから [.NET Framework 4.8] を選択し、[作成] を選択します。

   プロジェクトディレクトリを書き留めます。この例では、プロジェクトディレクトリは `C:\Users\Administrator\source\repos\eb-aspnet\eb-aspnet` です。
4. [新しい ASP.NET ウェブアプリケーションの作成] ダイアログが表示されます。[空] のテンプレートを選択します。次に、[作成] を選択します。

   この時点で、Visual Studio を使用して空の ASP.NET ウェブアプリケーションプロジェクトを作成しました。次に、ASP.NET ウェブアプリケーションのエントリポイントとして機能するウェブフォームを作成します。
5. [プロジェクト] メニューから、[新しいアイテムの追加] を選択します。[新しいアイテムの追加] ページで、[ウェブフォーム] を選択し、`Default.aspx` と名前を付けます。次に [追加] を選択します。
6. 以下を `Default.aspx:` に追加します

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
 Inherits="eb_aspnet.Default" %>

<!DOCTYPE html>

<html xmlns="https://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Hello Elastic Beanstalk!</title>
</head>
<body>
    <form id="body" runat="server">
        <div>
            Hello Elastic Beanstalk! This is an ASP.NET on Windows Server
 application.
        </div>
    </form>
```

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

ステップ 2: アプリケーションをローカルに実行する

Visual Studio で、[デバッグ] メニューから [デバッグの開始] を選択してアプリケーションをローカルで実行します。ページには「Hello Elastic Beanstalk!」と表示されます。これは ASP.NET on Windows Server アプリケーションです。

ステップ 3: を使用して ASP.NET アプリケーションをデプロイする AWS Toolkit for Visual Studio

Elastic Beanstalk 環境を作成し、そこに新しいアプリケーションをデプロイするには、以下の手順に従います。

環境を作成し、ASP.NET アプリケーションをデプロイするには

1. [ソリューションエクスプローラー] でアプリケーションを右クリックし、[AWS Elastic Beanstalkに公開] を選択します。
2. 新しい Elastic Beanstalk アプリケーションと環境の名前を選択します。
3. この時点以降は、Elastic Beanstalk に用意されているデフォルトを続行することも、どのオプションや設定でも好みに変更することもできます。
4. [確認] ページで、[デプロイ] を選択します。これにより、ASP.NET ウェブアプリケーションがパッケージ化され、Elastic Beanstalk にデプロイされます。

   Elastic Beanstalk が環境を作成するのに約 5 分かかります。Elastic Beanstalk デプロイ機能は、新しくデプロイされたコードで利用できるようになるまで、環境を監視します。[Env:**<environment name>**] タブには、環境のステータスが表示されます。

ステップ 4: Elastic Beanstalk でアプリケーションを実行する

環境を作成するプロセスが完了すると、[Env:**<environment name>**] タブに、アプリケーションを起動するドメイン URL など、環境とアプリケーションに関する情報が表示されます。このタブでこの URL を選択するか、ウェブブラウザにコピーして貼り付けます。

お疲れ様でした。Elastic Beanstalk で ASP.NET アプリケーションをデプロイしました。

ステップ 5：クリーンアップ

アプリケーションの使用が終了したら、 AWS Toolkit for Visual Studioで環境を終了できます。

環境を終了するには

1. [AWS Explorer] でElastic Beanstalk ノードとアプリケーションノードを展開します。アプリケーション環境を右クリックして、[環境の終了] を選択します。
2. プロンプトが表示されたら、[はい] を選択して、環境を終了することを確認します。Elastic Beanstalk が環境で実行されている AWS リソースを終了するまでに数分かかります。

AWS アプリケーションの リソース

1 つのインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありません。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。

- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

> **その他のチュートリアルを試す**
>
> 異なるアプリケーション例の他のチュートリアルを試したい場合は、「.NET Core on Windows の QuickStart」を参照してください。

1 つか 2 つのサンプルアプリケーションをデプロイし、ローカルで ASP.NET アプリケーションを開発して実行する準備が整ったら、「.NET 開発環境のセットアップ」を参照します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。

.NET 開発環境のセットアップ

このトピックでは、.NET 開発環境を設定し、アプリケーションを AWS Elastic Beanstalkにデプロイする前にローカルでテストする手順について説明します。また、便利なツールのインストール手順を提供するウェブサイトも参照します。

セクション

- IDE のインストール
- のインストール AWS Toolkit for Visual Studio

アプリケーション内から AWS リソースを管理する必要がある場合は、 AWS SDK for .NET をインストールします。例えば、Amazon S3 を使用してデータを保存し、取得できます。

AWS SDK for .NET を使用すると、Visual Studio プロジェクトテンプレート、 AWS .NET ライブラリ、C# コードサンプル、およびドキュメントを備えた 1 つのダウンロード可能なパッケージで数分で開始できます。ライブラリを使用したアプリケーションの構築方法については、C# による実用的な例が用意されています。ライブラリとコードサンプルの使用方法の学習に役立てるため、オンラインの動画チュートリアルとリファレンスドキュメントが用意されています。

詳細とインストール方法については、AWS SDK for .NET のホームページにアクセスしてください。

IDE のインストール

統合された開発環境（IDE）は、アプリケーション開発を用意にする幅広い機能を提供します。.NET 開発に IDE を使用していない場合は、最初に Visual Studio Community をお試しください。

Visual Studio コミュニティページにアクセスして、Visual Studio Community をダウンロードし、インストールします。

のインストール AWS Toolkit for Visual Studio

AWS Toolkit for Visual Studio は、開発者が を使用して .NET アプリケーションを簡単に開発、デバッグ、デプロイできるようにする Visual Studio IDE 用のオープンソースプラグインです AWS。インストール手順については、Toolkit for Visual Studio ホームページにアクセスしてください。

Elastic Beanstalk .NET Windows プラットフォームの使用

このトピックでは、Elastic Beanstalk で ASP.NET および .NET Core Windows ウェブアプリケーションを設定、ビルド、実行する方法について説明します。

AWS Elastic Beanstalk は、.NET プログラミングフレームワークと Windows Server のさまざまなバージョンのプラットフォームをサポートしています。完全なリストについては、AWS Elastic Beanstalk プラットフォームドキュメントの「IIS を使用する Windows Server での .NET」を参照してください。

Elastic Beanstalk には、Elastic Beanstalk 環境内の EC2 インスタンスで実行されるソフトウェアのカスタマイズに使用できる設定オプションが用意されています。アプリケーションの環境変数を設定し、Amazon S3 へのログローテーションを有効にし、.NET フレームワークを設定することができます。

設定オプションは実行中の環境の設定を変更するために Elastic Beanstalk コンソールで利用できます。環境を終了したときにその設定が失われないようにするため、保存済み設定を使用して設定を保存し、それを後で他の環境に適用することができます。

ソースコードの設定を保存する場合、設定ファイルを含めることができます。設定ファイルの設定は、環境を作成するたびに、またはアプリケーションをデプロイするたびに適用されます。設定ファイルを使用して、デプロイの間にパッケージをインストールしたり、スクリプトを実行したり、他のインスタンスのカスタマイズオペレーションを実行することもできます。

Elastic Beanstalk コンソールで適用される設定は、設定ファイルに同じ設定があれば、それらの設定を上書きします。これにより、設定ファイルでデフォルト設定を定義し、コンソールでそのデフォルト設定を環境固有の設定で上書きできます。設定の優先順位の詳細と設定の他の変更方法については、「」を参照してください設定オプション

Elastic Beanstalk コンソールでの.NET 環境の設定

Elastic Beanstalk コンソールでは、Amazon S3 のログローテーションを有効にしたり、アプリケーションが環境から読み取ることができる変数を設定したり、.NET フレームワーク設定を変更することができます。

Elastic Beanstalk コンソールで.NET 環境を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

コンテナオプション

- Target .NET runtime – CLR v2 を実行するには `2.0` に設定します。
- 32 ビットアプリケーションの有効化 – 32 ビットアプリケーションを実行するには `True` に設定します。

ログオプション

[Log Options (ログオプション)] セクションには、2 つの設定があります。

- インスタンスプロファイル – アプリケーションに関連付けられた Amazon S3 バケットへのアクセス許可が付与されているインスタンスプロファイルを指定します。
- [Enable log file rotation to Amazon S3] (Amazon S3 へのログファイルのローテーションの有効化) - アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーするかどうかを指定します。

環境プロパティ

環境プロパティ セクションでは、アプリケーションを実行している Amazon EC2 インスタンスの環境設定を指定できます。これらの設定は、キーと値のペアでアプリケーションに渡されます。`System.GetEnvironmentVariable` を使用して、それらのペアを読み取ります。同じキーが `web.config` に存在するとともに、環境プロパティとして存在する可能性があります。`System.Configuration` 名前空間を使用して、`web.config` から値を読み取ります。

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;
string endpoint = appConfig["API_ENDPOINT"];
```

詳細については、「環境変数およびその他のソフトウェア設定」を参照してください。

aws:elasticbeanstalk:container:dotnet:apppool 名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクを実行できます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

.NET プラットフォームは、.NET ランタイムの設定に使用できる `aws:elasticbeanstalk:container:dotnet:apppool` 名前空間のオプションを定義します。

以下の設定ファイルの例では、この名前空間で使用できる各オプションの設定を示しています。

Example .ebextensions/dotnet-settings.config

```
option_settings:
  aws:elasticbeanstalk:container:dotnet:apppool:
```

```
    Target Runtime: 2.0
    Enable 32-bit Applications: True
```

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

Elastic Beanstalk Windows サーバープラットフォームのメジャーバージョン間での移行

AWS Elastic Beanstalk には、Windows Server プラットフォームのメジャーバージョンがいくつかあります。このページでは、各メジャーバージョンでの主な改善点と、以降のバージョンに移行する前の考慮点について説明します。

Windows Server プラットフォームは現在バージョン 2 (v2) です。アプリケーションで v2 より前の Windows Server プラットフォームバージョンを使用している場合は、v2 に移行することをお勧めします。

Windows サーバープラットフォームのメジャーバージョンの最新情報

Windows サーバープラットフォーム V2

Elastic Beanstalk Windows Server プラットフォームのバージョン 2 (v2) が 2019 年 2 月にリリースされました。V2 では、いくつかの重要な点で Windows Server プラットフォームの動作が Elastic Beanstalk の Linux ベースのプラットフォームに近づいています。v2 は v1 と完全に下位互換性があり、v1 からの移行が容易です。

Windows Server プラットフォームでは次の機能がサポートされています。

- バージョニング – 各リリースでは新しいバージョン番号が取得され、環境の作成および管理時には、以前のバージョン (まだ使用可能) を参照できます。
- 拡張ヘルス – 詳細については、「Elastic Beanstalk でのヘルスレポートとモニタリングの強化」を参照してください。
- 変更不可の追加のバッチとローリングのデプロイ – デプロイポリシーの詳細については、「Elastic Beanstalk 環境へのアプリケーションのデプロイ」を参照してください。
- 変更不可の更新 – 更新タイプの詳細については、「設定変更」を参照してください。
- マネージドプラットフォーム更新 – 詳細については、「マネージドプラットフォーム更新」を参照してください。

> **Note**
>
> 新しいデプロイおよび更新機能は、拡張ヘルスに依存します。それらを使用するには、拡張ヘルスを有効にします。詳細については、「Elastic Beanstalk の拡張ヘルスレポートの有効化」を参照してください。

Windows サーバープラットフォーム V1

Elastic Beanstalk Windows Server プラットフォームのバージョン 1.0.0 (v1) が 2015 年 10 月にリリースされました。このバージョンでは、環境の作成および更新中に Elastic Beanstalk が設定ファイル内のコマンドを実行する順序が変更されています。

以前のプラットフォームのバージョンでは、ソリューションスタック名にバージョン番号が使用されていません。

- IIS 8.5 を実行する 64 ビット Windows Server 2012 R2
- IIS 8.5 を実行する 64 ビット Windows Server Core 2012 R2
- IIS 8 を実行する 64 ビット Windows Server 2012
- IIS 7.5 を実行する 64 ビット Windows Server 2008 R2

以前のバージョンでは、設定ファイルの処理順には一貫性がありません。環境の作成中に、アプリケーションソースが IIS にデプロイされると `Container Commands` が実行されます。実行中の環境へのデプロイ中は、新しいバージョンがデプロイされる前にコンテナコマンドが実行されます。スケールアップ中は、設定ファイルはまったく処理されません。

これに加えて、コンテナコマンドの実行前に IIS が開始されます。この動作により一部の顧客はコンテナコマンドに回避策を実装し、コマンドの完了後に再度 IIS を起動して開始するコマンドの前に IIS サーバーを一時停止することになります。

バージョン 1 ではこの一貫性のない状態が修正されており、Windows Server プラットフォームの動作が Elastic Beanstalk の Linux ベースのプラットフォームにより近いものになっています。v1 プラットフォームでは、Elastic Beanstalk は IIS サーバーを起動する前に常にコンテナコマンドを実行します。

v1 のプラットフォームソリューションスタックでは、Windows Server のバージョンの後に v1 が付きます。

- IIS 8.5 を実行する 64 ビット Windows Server 2012 R2 v1.1.0
- IIS 8.5 を実行する 64 ビット Windows Server Core 2012 R2 v1.1.0
- IIS 8 を実行する 64 ビット Windows Server 2012 v1.1.0
- IIS 7.5 を実行する 64 ビット Windows Server 2008 R2 v1.1.0

さらに、v1 プラットフォームはコンテナコマンドを実行する前に、アプリケーションソースバンドルのコンテンツを抽出して `C:\staging\` に保存します。コンテナコマンドの完了後に、このフォルダのコンテンツは .zip ファイルに圧縮され IIS にデプロイされます。このワークフローにより、コマンドまたはスクリプトを使用してアプリケーションソースバンドルのコンテンツを変更してからデプロイできます。

Windows サーバープラットフォームの以前のメジャーバージョンからの移行

環境を更新する前に、このセクションの移行の考慮事項についてお読みください。新しいバージョンに環境のプラットフォームを更新するには、「Elastic Beanstalk 環境のプラットフォームバージョンの更新」を参照してください。

V1 から V2 へ

Windows Server プラットフォーム v2 では、.NET Core 1.x および 2.0 をサポートしていません。アプリケーションを Windows Server v1 から v2 に移行中で、アプリケーションがこれらのいずれかの .NET Core バージョンを使用している場合は、v2 がサポートしている .NET Core バージョンにアプリケーションを更新します。サポートされているバージョンのリストについては、AWS Elastic Beanstalk プラットフォームの「IIS を使用する Windows Server での .NET」を参照してください。

アプリケーションでカスタム Amazon Machine Image (AMI) を使用している場合は、Windows Server プラットフォーム v2 AMI に基づいて新しいカスタム AMI を作成します。詳細については、「Elastic Beanstalk 環境でのカスタム Amazon マシンイメージ (AMI) の使用」を参照してください。

> **Note**
>
> Windows Server v2 の新しいデプロイおよび更新機能は、拡張ヘルスに依存します。環境を v2 に移行するときは、拡張ヘルスが無効になります。これらの機能を使用するには、拡張ヘルスを有効にします。詳細については、「Elastic Beanstalk の拡張ヘルスレポートの有効化」を参照してください。

V1 以前から

v1 からの移行の考慮点に加えて、v1 以前の Windows Server ソリューションスタックからアプリケーションを移行していて、現在コンテナコマンドを使用している場合は、新しいバージョンに移行するときに処理の不一致を回避するために追加したコマンドをすべて削除します。v1 からは、コンテナコマンドは、デプロイされるアプリケーションソース、および IIS が起動する前に、完全に実行することが保証されています。これにより、このステップ中に問題なく `C:\staging` のソースに変更を加え、IIS 設定ファイルを修正することができます。

例えば、 を使用して AWS CLI 、DLL ファイルを Amazon S3 からアプリケーションソースにダウンロードできます。

`.ebextensions\copy-dll.config`

```
container_commands:
  copy-dll:
    command: aws s3 cp s3://amzn-s3-demo-bucket/dlls/large-dll.dll .\lib\
```

設定ファイルの使用の詳細については、「設定ファイル (`.ebextensions`) による高度な環境のカスタマイズ」を参照してください。

デプロイマニフェストを使用した複数のアプリケーションと ASP.NET コアアプリケーションの実行

デプロイマニフェストで、Elastic Beanstalk にアプリケーションをデプロイする方法を伝えることができます。この方法を使用すると、ウェブサイトのルートパスで実行される個々の ASP.NET アプリケーション用のソースバンドルを生成するために、`MSDeploy` を使用する必要がなくなります。マニフェストファイルを使用することにより、異なるパスで複数のアプリケーションを実行することが可能になります。または、Elastic Beanstalk に対して、ASP.NET Core を使用してアプリケーションのデプロイと実行を行うように指示することもできます。また、デプロイマニフェストは、アプリケーションを実行するためのアプリケーションプールの設定にも使用できます。

デプロイマニフェストは、Elastic Beanstalk へ .NET Core アプリケーションのサポートを追加します。.NET Framework アプリケーションは、デプロイマニフェストなしでデプロイできます。ただし、.NET Core アプリケーションを Elastic Beanstalk で実行する場合は、デプロイマニフェストが必要です。デプロイマニフェストを使用する際には、まず各アプリケーションのサイトアーカイブを作成します。その上で、デプロイマニフェストを集録するための別の ZIP アーカイブ内に、先に作成したサイトアーカイブをバンドルします。

デプロイマニフェストを使用すると、異なるパスで複数のアプリケーションを実行することもできます。デプロイマニフェストは、多くのデプロイのターゲットを、サイトアーカイブと IIS が実行するパスにより個別に定義します。たとえば、`/api` パスで非同期リクエストを行うウェブ API を実行し、ルートパスでその API を実行するウェブアプリケーションを実行できます。

デプロイマニフェストを使用して、カスタムバインディングと物理パスを使用して IIS ウェブサイトを設定できます。これにより、アプリケーションをデプロイする前に、特定のポートまたはホスト名をリッスンするウェブサイトを設定できます。

デプロイマニフェストは、IIS または Kestrel のアプリケーションプールを使用して、複数のアプリケーションを実行するために使用することもできます。アプリケーションプールを設定して、アプリケーションの定期的な再起動、32 ビットアプリケーションの実行、または、.NET Framework ランタイムの特定バージョンの使用ができます。

完全にカスタマイズするには、Windows PowerShell で独自のデプロイスクリプトを作成し、Elastic Beanstalk にアプリケーションのインストール、アンインストール、再起動のために実行するスクリプトを伝えます。

デプロイマニフェストと関連機能を使用するには、Windows Server プラットフォームバージョン 1.2.0 以降が必要です。

IIS リセットのスキップなど、使用可能なすべての設定オプション、プロパティ、高度な機能の詳細については、デプロイマニフェストスキーマリファレンスを参照してください。

セクション

- .NET core アプリケーション
- 複数のアプリケーションを実行する
- IIS ウェブサイトを設定する
- アプリケーションリクエストルーティング (ARR) の使用
- アプリケーションプールを設定する
- カスタムデプロイを定義する
- デプロイマニフェストスキーマリファレンス

.NET core アプリケーション

デプロイマニフェストを使用すると、Elastic Beanstalk で .NET Core アプリケーションを実行することができます。.NET Core は .NET のクロスプラットフォームバージョンで、コマンドラインツー

ル (dotnet) が付属しています。このマニフェストにより、アプリケーションの生成、ローカルでの実行、および公開の準備を行うことができます。

Elastic Beanstalk で .NET Core アプリケーションを実行するには、`dotnet publish` を実行し、その出力からディレクトリをすべて削除した内容を、ZIP アーカイブにパッケージします。サイトアーカイブを、デプロイマニフェストを使って、デプロイターゲットのタイプ `aspNetCoreWeb` と一緒にソースバンドルに配置します。

以下のデプロイマニフェストは、ルートパスの `dotnet-core-app.zip` という名前のサイトアーカイブから .NET Core アプリケーションを実行します。

Example aws-windows-deployment-manifest.json - .NET core

```
{
  "manifestVersion": 1,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "my-dotnet-core-app",
        "parameters": {
          "archive": "dotnet-core-app.zip",
          "iisPath": "/"
        }
      }
    ]
  }
}
```

マニフェストとサイトアーカイブを ZIP アーカイブにバンドルし、ソースバンドルを作成します。

Example dotnet-core-bundle.zip

```
.
|-- aws-windows-deployment-manifest.json
`-- dotnet-core-app.zip
```

サイトアーカイブには、コンパイルされたアプリケーションコード、依存関係、`web.config` ファイルが含まれています。

Example dotnet-core-app.zip

```
.
```

```
|-- Microsoft.AspNetCore.Hosting.Abstractions.dll
|-- Microsoft.AspNetCore.Hosting.Server.Abstractions.dll
|-- Microsoft.AspNetCore.Hosting.dll
|-- Microsoft.AspNetCore.Http.Abstractions.dll
|-- Microsoft.AspNetCore.Http.Extensions.dll
|-- Microsoft.AspNetCore.Http.Features.dll
|-- Microsoft.AspNetCore.Http.dll
|-- Microsoft.AspNetCore.HttpOverrides.dll
|-- Microsoft.AspNetCore.Server.IISIntegration.dll
|-- Microsoft.AspNetCore.Server.Kestrel.dll
|-- Microsoft.AspNetCore.WebUtilities.dll
|-- Microsoft.Extensions.Configuration.Abstractions.dll
|-- Microsoft.Extensions.Configuration.EnvironmentVariables.dll
|-- Microsoft.Extensions.Configuration.dll
|-- Microsoft.Extensions.DependencyInjection.Abstractions.dll
|-- Microsoft.Extensions.DependencyInjection.dll
|-- Microsoft.Extensions.FileProviders.Abstractions.dll
|-- Microsoft.Extensions.FileProviders.Physical.dll
|-- Microsoft.Extensions.FileSystemGlobbing.dll
|-- Microsoft.Extensions.Logging.Abstractions.dll
|-- Microsoft.Extensions.Logging.dll
|-- Microsoft.Extensions.ObjectPool.dll
|-- Microsoft.Extensions.Options.dll
|-- Microsoft.Extensions.PlatformAbstractions.dll
|-- Microsoft.Extensions.Primitives.dll
|-- Microsoft.Net.Http.Headers.dll
|-- System.Diagnostics.Contracts.dll
|-- System.Net.WebSockets.dll
|-- System.Text.Encodings.Web.dll
|-- dotnet-core-app.deps.json
|-- dotnet-core-app.dll
|-- dotnet-core-app.pdb
|-- dotnet-core-app.runtimeconfig.json
`-- web.config
```

複数のアプリケーションを実行する

デプロイマニフェストを使って、複数のデプロイターゲットを指定することにより、複数のアプリケーションを実行できます。

以下のデプロイマニフェストでは 2 つの .NET Core アプリケーションを設定します。`WebApiSampleApp` アプリケーションは、シンプルなウェブ API を実装し、`/api` パスで非同

期リクエストを提供します。DotNetSampleApp アプリケーションは、ルートパスでリクエストを処理するウェブアプリケーションです。

Example aws-windows-deployment-manifest.json - multiple apps

```
{
  "manifestVersion": 1,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "WebAPISample",
        "parameters": {
          "appBundle": "WebApiSampleApp.zip",
          "iisPath": "/api"
        }
      },
      {
        "name": "DotNetSample",
        "parameters": {
          "appBundle": "DotNetSampleApp.zip",
          "iisPath": "/"
        }
      }
    ]
  }
}
```

複数のアプリケーションのサンプルアプリケーションはこちらで入手できます:

- デプロイ可能なソースバンドル - dotnet-multiapp-sample-bundle-v2.zip
- ソースコード - dotnet-multiapp-sample-source-v2.zip

IIS ウェブサイトを設定する

デプロイマニフェストを使用して、IIS ウェブサイトにカスタムバインディングと物理パスを設定できます。これは、特定のポートをリッスンしたり、カスタムホスト名を使用したり、特定のディレクトリのコンテンツを提供するウェブサイトを設定する必要がある場合に便利です。

次のデプロイマニフェストは、特定のポート番号とカスタム物理パスを使用して HTTP をリッスンするカスタム IIS ウェブサイトを設定します。

Example aws-windows-deployment-manifest.json - IIS ウェブサイト設定

```
{
  "manifestVersion": 1,
  "iisConfig": {
    "websites": [
      {
        "name": "MyCustomSite",
        "physicalPath": "C:\inetpub\wwwroot\mysite",
        "bindings": [
          {
            "protocol": "http",
            "port": 8080,
            "hostName": "mysite.local"
          }
        ]
      }
    ]
  },
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "my-dotnet-core-app",
        "parameters": {
          "appBundle": "dotnet-core-app.zip",
          "iisWebSite": "MyCustomSite",
          "iisPath": "/"
        }
      }
    ]
  }
}
```

この例では、以下のようになっています：

- 「MyCustomSite」という名前のウェブサイトがカスタム物理パスで作成されます
- ウェブサイトには、特定のホスト名を持つポート 8080 に HTTP バインディングがあります。
- ASP.NET Core アプリケーションは、 `iisWebSite`パラメータを使用してこのカスタムウェブサイトにデプロイされます。

アプリケーションリクエストルーティング (ARR) の使用

Application Request Routing (ARR) モジュールと URL Rewrite モジュールはプリインストールされており、Elastic Beanstalk Windows AMIs で使用できます。これらのモジュールにより、ebextensions またはアプリケーション設定を使用した IIS 設定による高度なルーティングシナリオと URL 操作が可能になります。

次の例は、カスタムポートを使用してウェブサイトを設定するシンプルなデプロイマニフェストと、基本的な ARR ルーティングを設定する ebextensions 設定を示しています。

Example aws-windows-deployment-manifest.json - シンプルな ARR 設定

```
{
  "manifestVersion": 1,
  "iisConfig": {
    "websites": [
      {
        "name": "ARRSite",
        "physicalPath": "C:\\inetpub\\wwwroot\\arrsite",
        "bindings": [
          {
            "protocol": "http",
            "port": 8080,
            "hostName": "localhost"
          }
        ]
      }
    ]
  },
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "BackendApp",
        "parameters": {
          "appBundle": "backend-app.zip",
          "iisWebSite": "ARRSite",
          "iisPath": "/backend"
        }
      }
    ]
  }
}
```

ARR 設定は ebextensions を介して行われます。次の設定は、基本的な ARR ルーティングルールを設定します。

Example .ebextensions/arr-config.config - 基本的な ARR 設定

```
files:
  "C:\\temp\\configure-arr.ps1":
    content: |
      # Enable ARR proxy at server level
      Set-WebConfigurationProperty -PSPath 'MACHINE/WEBROOT/APPHOST' -Filter
 'system.webServer/proxy' -Name 'enabled' -Value 'True'

      # Clear any existing global rules to avoid conflicts
      Clear-WebConfiguration -PSPath 'MACHINE/WEBROOT/APPHOST' -Filter
 'system.webServer/rewrite/globalRules'

      # Add global rule to route all requests to backend
      Add-WebConfigurationProperty -PSPath 'MACHINE/WEBROOT/APPHOST' `
        -Filter 'system.webServer/rewrite/globalRules' `
        -Name '.' `
        -Value @{
          name = 'Route_to_Backend'
          stopProcessing = 'True'
          match = @{ url = '^(?!backend/)(.*)' }
          action = @{
            type = 'Rewrite'
            url = 'http://localhost:8080/backend/{R:1}'
          }
        }

container_commands:
  01_configure_arr:
    command: powershell -ExecutionPolicy Bypass -File "C:\\temp\\configure-arr.ps1"
    waitAfterCompletion: 0
```

この設定では、ポート 8080 にウェブサイトを作成し、すべての受信リクエストをそのサイトで実行されているバックエンドアプリケーションにルーティングするように ARR を設定します。

アプリケーションプールを設定する

Windows 環境では、複数のアプリケーションをサポートできます。そのためには、次のような 2 つのアプローチが存在します。

- Kestrel ウェブサーバーでは、アウトプロセスホスティングモデルを使用できます。このモデルでは、複数のアプリケーションを 1 つのアプリケーションプールで実行するように構成します。
- インプロセスホスティングモデルを使用する場合には、複数のアプリケーションを実行するために複数のアプリケーションプールを用意して、各プール内では単一アプリケーションのみを実行します。IIS サーバーを使用していて、複数のアプリケーションを実行したい場合には、この方法を使用する必要があります。

1 つのアプリケーションプールで複数のアプリケーションを実行するように Kestrel を構成するには、`hostingModel="OutofProcess"` ファイルに `web.config` を追加で記述します。次に挙げるサンプルを参考にしてください。

Example web.config - Kestrel アウトプロセスホスティングモデル用

```
<configuration>
<location path="." inheritInChildApplications="false">
<system.webServer>
<handlers>
<add
    name="aspNetCore"
    path="*" verb="*"
    modules="AspNetCoreModuleV2"
    resourceType="Unspecified" />
</handlers>
<aspNetCore
    processPath="dotnet"
    arguments=".\CoreWebApp-5-0.dll"
    stdoutLogEnabled="false"
    stdoutLogFile=".\logs\stdout"
    hostingModel="OutofProcess" />
</system.webServer>
</location>
</configuration>
```

Example aws-windows-deployment-manifest.json - 複数のアプリケーション

```
{
"manifestVersion": 1,
  "deployments": {"msDeploy": [
      {"name": "Web-app1",
        "parameters": {"archive": "site1.zip",
```

```
          "iisPath": "/"
        }
      },
      {"name": "Web-app2",
        "parameters": {"archive": "site2.zip",
          "iisPath": "/app2"
        }
      }
    ]
  }
}
```

IIS では、インプロセスホスティングモデルが使用されているため、1 つのアプリケーションプールでの複数のアプリケーションの実行はサポートされていません。したがって、複数のアプリケーションを構成するには、各アプリケーションを個別のアプリケーションプールに割り当てる必要があります。つまり、1 つのアプリケーションプールに割り当てられるのは、1 つのアプリケーションのみです。

`aws-windows-deployment-manifest.json`ファイルで、IIS を異なるアプリケーションプールを使用するように構成できます。次のサンプルファイルを参考にしながら、同様な更新を行ってください。

- `iisConfig` というサブセクションを含む、`appPools` というセクションを追加します。
- `appPools` ブロックに、アプリケーションプールをリストします。
- `deployments` セクションで、各アプリケーションの `parameters` セクションを定義します。
- `parameters` セクションでは、各アプリケーションのアーカイブ、実行するパス、および実行するために使用する `appPool` を指定します。

次のデプロイマニフェストでは、10 分ごとにアプリケーションを再起動する 2 つのアプリケーションプールを構成しています。同時に、指定されたパスで実行される .NET Framework ウェブアプリケーションにも、アプリケーションをアタッチしています。

Example aws-windows-deployment-manifest.json - アプリケーションプールごとに 1 つのアプリケーション

```
{
"manifestVersion": 1,
  "iisConfig": {"appPools": [
      {"name": "MyFirstPool",
```

```
        "recycling": {"regularTimeInterval": 10}
      },
      {"name": "MySecondPool",
       "recycling": {"regularTimeInterval": 10}
      }
     ]
    },
  "deployments": {"msDeploy": [
      {"name": "Web-app1",
        "parameters": {
           "archive": "site1.zip",
           "iisPath": "/",
           "appPool": "MyFirstPool"
           }
      },
      {"name": "Web-app2",
        "parameters": {
           "archive": "site2.zip",
           "iisPath": "/app2",
           "appPool": "MySecondPool"
          }
      }
     ]
    }
}
```

カスタムデプロイを定義する

さらにきめ細かく制御するには、カスタムデプロイを定義することによって、アプリケーションのデプロイを完全にカスタマイズできます。

このデプロイマニフェストは、32 ビットモードで PowerShell スクリプトを実行するように Elastic Beanstalk に指示します。3 つのスクリプトを指定します。1 つはインスタンスの起動とデプロイ中に実行される`install`スクリプト (`siteInstall.ps1`) 、もう 1 つはデプロイ中に新しいバージョンをインストールする前に実行される`uninstall`スクリプト (`siteUninstall.ps1`) 、もう 1 つは AWS 管理コンソールでアプリサーバーの再起動を選択したときに実行される`restart`スクリプト (`siteRestart.ps1`) です。

Example aws-windows-deployment-manifest.json - custom deployment

```
{
  "manifestVersion": 1,
```

```
  "deployments": {
    "custom": [
      {
        "name": "Custom site",
        "architecture" : 32,
        "scripts": {
          "install": {
            "file": "siteInstall.ps1"
          },
          "restart": {
            "file": "siteRestart.ps1"
          },
          "uninstall": {
            "file": "siteUninstall.ps1"
          }
        }
      }
    ]
  }
}
```

ソースバンドルには、マニフェスト、スクリプトと一緒にアプリケーションの実行に必要なアーティファクトをすべて含めます。

Example Custom-site-bundle.zip

```
.
|-- aws-windows-deployment-manifest.json
|-- siteInstall.ps1
|-- siteRestart.ps1
|-- siteUninstall.ps1
`-- site-contents.zip
```

デプロイマニフェストスキーマリファレンス

デプロイマニフェストは、Elastic Beanstalk が Windows アプリケーションをデプロイおよび設定する方法を定義する JSON ファイルです。このセクションでは、マニフェストスキーマでサポートされているすべてのプロパティと設定オプションの包括的なリファレンスを提供します。

マニフェスト構造

デプロイマニフェストは、次の最上位構造を持つ特定の JSON スキーマに従います。

Example 基本的なマニフェスト構造

```
{
  "manifestVersion": 1,
  "skipIISReset": false,
  "iisConfig": {
    "websites": [...],
    "appPools": [...]
  },
  "deployments": {
    "msDeploy": [...],
    "aspNetCoreWeb": [...],
    "custom": [...]
  }
}
```

最上位プロパティ

`manifestVersion` (必須)

タイプ: 数値

デフォルト: 1

有効な値： 1

マニフェストスキーマのバージョンを指定します。現在、バージョン 1 のみがサポートされています。

`skipIISReset` (オプション)

型: ブール

デフォルト: false

アプリケーションのデプロイ中に IIS をリセットするかどうかを制御します。このフラグは、`msDeploy`と の両方の`aspNetCoreWeb`デプロイタイプに影響します。

動作：

- 指定なしまたは *`false`* (デフォルト): IIS リセットは、インストール、アンインストール、および更新オペレーション中に実行されます。これは従来の動作です。
- *`true`*： IIS リセットはデプロイオペレーション中にスキップされます。

利点：

- ダウンタイムの短縮 – アプリケーションでは、デプロイ中のサービスの中断が短くなります。
- デプロイの高速化 — IIS が完全に再起動して再初期化するために必要な時間を排除します。

> **Note**
>
> を使用する場合`skipIISReset`、RestartAppServer オペレーションは、このフラグ設定に関係なく IIS リセットを実行します。

例:

```
{
  "manifestVersion": 1,
  "skipIISReset": true,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "my-dotnet-core-app",
        "parameters": {
          "archive": "dotnet-core-app.zip",
          "iisPath": "/"
        }
      }
    ]
  }
}
```

`deployments` (必須)

タイプ: オブジェクト

アプリケーションのデプロイ設定が含まれます。このオブジェクトには`msDeploy`、、`aspNetCoreWeb`、および のcustomデプロイタイプを含めることができます。

`iisConfig` (オプション)

タイプ: オブジェクト

アプリケーションをデプロイする前に適用する IIS 構成設定を定義します。ウェブサイトとアプリケーションプールの両方の設定をサポートします。

IIS 設定

`iisConfig` セクションでは、アプリケーションをデプロイする前に IIS 設定を構成することができます。これには、特定の設定でアプリケーションプールを設定し、カスタムバインディングで IIS ウェブサイトを設定することが含まれます。

IIS ウェブサイト

IIS ウェブサイトでは、アプリケーションをデプロイする前に、物理パスやネットワークバインディングなどのカスタムウェブサイト設定を行うことができます。

> さまざまな IIS ウェブサイトを作成するための重要な考慮事項
>
> - ウェブサイトのセットアップ順序： ウェブサイトは、`websites`配列に表示される順序で順番に設定されます。プラットフォームは各ウェブサイト設定を順番に処理するため、ウェブサイト間に依存関係がある場合は、適切な順序にします。
> - ファイアウォールとポートへのアクセス： ポート 80 のみがデフォルトの Elastic Beanstalk Windows ファイアウォール設定で自動的に公開されます。非標準ポートを使用するようにウェブサイトを設定する場合は、ebextensions またはカスタムデプロイスクリプトを使用してカスタムファイアウォールルールを定義し、これらのポートへの外部アクセスを許可する必要があります。

Example ウェブサイト設定

```
{
  "iisConfig": {
    "websites": [
      {
        "name": "MyCustomSite",
        "physicalPath": "C:\inetpub\wwwroot\mysite",
        "bindings": [
          {
            "protocol": "http",
            "port": 8080,
            "hostName": "mysite.local"
          },
          {
            "protocol": "https",
            "port": 8443
```

```
          }
        ]
      }
    ]
  }
}
```

ウェブサイトのプロパティ

`name` (必須)

タイプ: 文字列

IIS ウェブサイトの名前。この名前は IIS Manager のウェブサイトを識別するために使用され、IIS 設定内で一意である必要があります。

`physicalPath` (必須)

タイプ: 文字列

ウェブサイトファイルが保存されているサーバーの物理パス。このパスは IIS ワーカープロセスにアクセスできる必要があります。

`bindings` (必須)

型: 配列

最小項目： 1

ウェブサイトがネットワークリクエストにどのように応答するかを定義するバインディング設定の配列。各バインディングは、プロトコル、ポート、およびオプションのホスト名を指定します。

ウェブサイトバインディング

ウェブサイトバインディングは、IIS ウェブサイトが受信リクエストをリッスンするネットワークエンドポイントを定義します。

`protocol` (必須)

タイプ: 文字列

有効な値： "http"、"https"

バインディングに使用されるプロトコル。

`port` (必須)

タイプ: 整数

有効範囲： 1 ~ 65535

ウェブサイトがリクエストをリッスンするポート番号。

`hostName` (オプション)

タイプ: 文字列

バインディングのホスト名 (ドメイン名) 。

アプリケーションプール

アプリケーションプールはアプリケーション間の分離を提供し、アプリケーションのグループのランタイム設定を構成することができます。

Example アプリケーションプールの設定

```
{
  "iisConfig": {
    "appPools": [
      {
        "name": "MyAppPool",
        "enable32Bit": false,
        "managedPipelineMode": "Integrated",
        "managedRuntimeVersion": "v4.0",
        "queueLength": 1000,
        "cpu": {
          "limitPercentage": 80,
          "limitAction": "Throttle",
          "limitMonitoringInterval": 5
        },
        "recycling": {
          "regularTimeInterval": 1440,
          "requestLimit": 10000,
          "memory": 1048576,
          "privateMemory": 524288
        }
      }
```

```
    ]
  }
}
```

アプリケーションプールのプロパティ

`name` (必須)

タイプ: 文字列

アプリケーションプールの名前。この名前は、デプロイ設定でプールを参照するために使用されます。

`enable32Bit` (オプション)

型: ブール

32 ビットアプリケーションが 64 ビットバージョンの Windows で実行できるようにします。32 ビットの互換性を必要とするレガシーアプリケーション`true`の場合は、 に設定します。

`managedPipelineMode` (オプション)

タイプ: 文字列

有効な値： "Integrated"、"Classic"

アプリケーションプールのリクエスト処理モードを指定します。

`managedRuntimeVersion` (オプション)

タイプ: 文字列

有効な値：「マネージドコードなし」、「v2.0」、「v4.0」

アプリケーションプールの .NET Framework バージョンを指定します。

`queueLength` (オプション)

タイプ: 整数

追加のリクエストを拒否する前に HTTP.sys がアプリケーションプールに対してキューに入れるリクエストの最大数。

CPU 設定

`cpu` オブジェクトは、アプリケーションプールの CPU 使用率制限とモニタリングを設定します。

`limitPercentage` (オプション)

タイプ: 数値

アプリケーションプール内のワーカープロセスが消費できる CPU 時間の最大パーセンテージ。

`limitAction` (オプション)

タイプ: 文字列

有効な値： "NoAction"、"KillW3wp"、"Throttle"、"ThrottleUnderLoad"

CPU 制限に達したときに実行するアクション。

`limitMonitoringInterval` (オプション)

タイプ: 数値

CPU モニタリングとスロットリングの制限のリセット期間 (分単位) 。

リサイクル設定

`recycling` オブジェクトは、アプリケーションプールワーカープロセスをいつどのようにリサイクルするかを設定します。

`regularTimeInterval` (オプション)

タイプ: 整数

アプリケーションプールがリサイクルされるまでの時間間隔 (分単位) 。時間ベースのリサイクルを無効にするには、0 に設定します。

`requestLimit` (オプション)

タイプ: 整数

リサイクル前にアプリケーションプールが処理するリクエストの最大数。

`memory` (オプション)

タイプ: 整数

ワーカープロセスのリサイクルをトリガーする仮想メモリの量 (キロバイト単位) 。

`privateMemory` (オプション)

タイプ: 整数

ワーカープロセスのリサイクルをトリガーするプライベートメモリの量 (キロバイト単位) 。

デプロイタイプ

`deployments` オブジェクトには、さまざまなアプリケーションタイプのデプロイ設定の配列が含まれています。各デプロイタイプには、特定のプロパティとユースケースがあります。

MSDeploy デプロイ

MSDeploy デプロイは、Web Deploy (MSDeploy) を使用してデプロイできる従来の .NET Framework アプリケーションに使用されます。

Example MSDeploy デプロイ設定

```
{
  "deployments": {
    "msDeploy": [
      {
        "name": "WebApp",
        "description": "Main web application",
        "parameters": {
          "appBundle": "webapp.zip",
          "iisPath": "/",
          "appPool": "DefaultAppPool"
        }
      }
    ]
  }
}
```

MSDeploy デプロイプロパティ

`name` (必須)

タイプ: 文字列

デプロイの一意の名前。この名前は、マニフェスト内のすべてのデプロイで一意である必要があります。

`description` (オプション)

タイプ: 文字列

デプロイの人間が読める説明。

`parameters` (必須)

タイプ: オブジェクト

MSDeploy オペレーションの設定パラメータ。

`scripts` (オプション)

タイプ: オブジェクト

デプロイライフサイクルのさまざまな段階で実行する PowerShell スクリプト。

MSDeploy パラメータ

`appBundle` (必須)

タイプ: 文字列

マニフェストファイルに対するアプリケーションバンドル (ZIP ファイル) へのパス。このバンドルには、デプロイするアプリケーションファイルが含まれています。

`iisWebSite` (オプション)

タイプ: 文字列

デフォルト： "Default Web Site"

アプリケーションをデプロイする IIS ウェブサイト。デフォルトでは、アプリケーションは「デフォルトウェブサイト」にデプロイされます。必要に応じて、 `iisConfig.websites`セクションで設定されたウェブサイト名など、別のウェブサイト名を指定できます。

`iisPath` (オプション)

タイプ: 文字列

デフォルト： "/"

アプリケーションがデプロイされる IIS の仮想ディレクトリパス。ルートパスには「/」、サブディレクトリには「/api」を使用します。

`appPool` (オプション)

タイプ: 文字列

このアプリケーションを実行するアプリケーションプールの名前。

ASP.NET Core デプロイ

ASP.NET Core デプロイは、.NET Core および .NET 5+ アプリケーション専用に設計されています。

Example ASP.NET Core デプロイ設定

```
{
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "CoreAPI",
        "description": "ASP.NET Core Web API",
        "parameters": {
          "appBundle": "coreapi.zip",
          "iisPath": "/api",
          "appPool": "CoreAppPool"
        }
      }
    ]
  }
}
```

ASP.NET Core デプロイでは、MSDeploy デプロイと同じプロパティ構造を使用します。主な違いは、アプリケーションに使用されるランタイム環境とホスティングモデルです。

ASP.NET Core デプロイパラメータ

`appBundle` (必須)

タイプ: 文字列

マニフェストファイルに対するアプリケーションバンドルへのパス。これは、ZIP アーカイブでも、公開された ASP.NET Core アプリケーションを含むディレクトリパスでもかまいません。

`iisWebSite` (オプション)

タイプ: 文字列

デフォルト： "Default Web Site"

ASP.NET Core アプリケーションをデプロイする IIS ウェブサイト。デフォルトでは、アプリケーションは「デフォルトウェブサイト」にデプロイされます。必要に応じて、`iisConfig.websites`セクションで設定されたウェブサイト名など、別のウェブサイト名を指定できます。

`iisPath` (オプション)

タイプ: 文字列

デフォルト： "/"

ASP.NET Core アプリケーションの IIS の仮想ディレクトリパス。

`appPool` (オプション)

タイプ: 文字列

ASP.NET Core アプリケーションのアプリケーションプール。プールは、ASP.NET Core ホスティング用に適切に設定されます。

カスタムデプロイ

カスタムデプロイでは、PowerShell スクリプトを使用してデプロイプロセスを完全に制御できます。このデプロイタイプは、カスタムインストール、設定、またはデプロイロジックを必要とする複雑なシナリオに役立ちます。

Example カスタムデプロイ設定

```
{
  "deployments": {
    "custom": [
      {
        "name": "CustomService",
        "description": "Custom Windows service deployment",
        "architecture": 32,
        "scripts": {
          "install": {
            "file": "install-service.ps1"
          },
          "restart": {
            "file": "restart-service.ps1"
          },
          "uninstall": {
```

```
            "file": "uninstall-service.ps1",
            "ignoreErrors": true
          }
        }
      }
    ]
  }
}
```

カスタムデプロイプロパティ

`name` (必須)

タイプ: 文字列

カスタムデプロイの一意の名前。

`description` (オプション)

タイプ: 文字列

カスタムデプロイの説明。

`architecture` (オプション)

タイプ: 整数

デフォルト： 32

有効な値： 32、64

Powershell スクリプトの実行モードのアーキテクチャ仕様

`scripts` (必須)

タイプ: オブジェクト

デプロイ動作を定義する PowerShell スクリプト。カスタムデプロイでは、他のデプロイタイプと比較して追加のスクリプトタイプがサポートされています。

デプロイスクリプト

デプロイスクリプトは、デプロイライフサイクル中の特定の時点で実行される PowerShell スクリプトです。異なるデプロイタイプは、異なるスクリプトイベントのセットをサポートします。

スクリプトイベント

デプロイタイプに応じて、次のスクリプトイベントを使用できます。

標準デプロイスクリプト (msDeploy および aspNetCoreWeb)

`preInstall`

アプリケーションがインストールまたは更新される前に実行されます。

`postInstall`

アプリケーションのインストールまたは更新後に実行されます。

`preRestart`

アプリケーションを再起動する前に を実行します。

`postRestart`

アプリケーションが再起動された後に実行されます。

`preUninstall`

アプリケーションがアンインストールされる前に を実行します。

`postUninstall`

アプリケーションがアンインストールされた後に実行されます。

カスタムデプロイスクリプト (カスタムデプロイのみ)

`install`

カスタムデプロイ用のプライマリインストールスクリプト。このスクリプトは、アプリケーションまたはサービスのインストールを担当します。

`restart`

アプリケーションまたはサービスを再起動するスクリプト。環境が再起動されると呼び出されます。

`uninstall`

アプリケーションまたはサービスをアンインストールするスクリプト。環境の終了またはアプリケーションの削除中に呼び出されます。

スクリプトプロパティ

各スクリプトは、次のプロパティを持つオブジェクトとして定義されます。

`file` (必須)

タイプ: 文字列

マニフェストファイルに対する PowerShell スクリプトファイルへのパス。スクリプトには`.ps1`拡張子が必要です。

`ignoreErrors` (オプション)

型: ブール

デフォルト: false

に設定すると`true`、スクリプトが失敗してもデプロイは続行されます。これは、重要ではないスクリプトまたはクリーンアップオペレーションに使用します。

Example スクリプト設定の例

```
{
  "scripts": {
    "preInstall": {
      "file": "backup-config.ps1",
      "ignoreErrors": true
    },
    "postInstall": {
      "file": "configure-app.ps1"
    }
  }
}
```

Windows プラットフォームブランチでの EC2 Fast Launch の使用

EC2 Fast Launch 機能は、Elastic Beanstalk 環境での Windows インスタンスの起動時間を短縮します。このトピックでは、Elastic Beanstalk 環境でこの機能を使用する方法について説明します。2025 年 1 月 22 日にリリースされた Windows プラットフォームバージョン 2.16.2 以降、Elastic Beanstalk プラットフォームリリースには、EC2 Fast Launch が有効になっている基本 AMIs が含まれています。

デフォルトの EC2 Fast Launch の可用性

最新の Elastic Beanstalk Windows プラットフォームバージョンには、EC2 Fast Launch AMIs が含まれており、追加コストはかかりません。ただし、新しいプラットフォームバージョンがリリースされると、EC2 Fast Launch は古いプラットフォームバージョンのベース AMIs で自動的に有効にならない場合があります。

EC2 Fast Launch が自動的に有効になっている基本 AMIs を使用するには、最新の Windows プラットフォームバージョンにアップグレードすることをお勧めします。ただし、既存のプラットフォームバージョンを引き続き使用する必要がある場合は、環境のベース AMI で EC2 Fast Launch を手動で有効にできます。手順については、EC2 Fast Launch の手動設定 を参照してください。

EC2 Fast Launch の手動設定

> **Note**
>
> EC2 Fast Launch を手動で有効にすると、EC2 Fast Launch が自動的に有効になっているプラットフォームバージョンを使用する場合と比較して、追加コストが発生する可能性があります。EC2 Fast Launch のコストの詳細については、「Amazon EC2 ユーザーガイド」の「EC2 Fast Launch の基盤となるリソースのコストを管理する」ページを参照してください。 Amazon EC2

Elastic Beanstalk 環境で使用される Windows ベース AMI で EC2 Fast Launch を有効にするには、次の手順に従います。

Elastic Beanstalk 環境で EC2 Fast Launch を手動で有効にするには

1. 環境のベース AMI を特定します。

   「カスタム AMI の作成」の手順に従って、環境のベース AMI ID を特定します。カスタム AMI を作成する必要はありません。現在のベース AMI ID を見つけるステップに従うだけで済みます。

2. AMI で EC2 Fast Launch を有効にします。

   AMI の EC2 Fast Launch を設定するには、Amazon EC2 ユーザーガイド」のEC2 Fast Launch を有効にする」の手順を使用します。

.NET アプリケーション環境に Amazon RDS DB インスタンスを追加

このトピックでは、Elastic Beanstalk コンソールを使用して Amazon RDS を作成する手順について説明します。Amazon Relational Database Service (Amazon RDS) DB インスタンスを使用して、アプリケーションによって収集および変更されたデータを保存することができます。データベースを環境に結合して Elastic Beanstalk で管理することも、分離したものとして作成して別のサービスで外部的に管理することもできます。これらの手順では、データベースは環境に結合され、Elastic Beanstalk によって管理されます。Amazon RDS と Elastic Beanstalk の統合の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

セクション

- 環境に DB インスタンスを追加
- ドライバのダウンロード
- データベースへの接続

環境に DB インスタンスを追加

お客様の環境に DB インスタンスを追加するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. DB エンジンを選択して、ユーザー名とパスワードを入力します。
6. ページの最下部で [適用] を選択し変更を保存します。

DB インスタンスの追加には約 10 分かかります。環境の更新が完了すると、DB インスタンスのホスト名とその他の接続情報は以下の環境プロパティを通じてアプリケーションに使用できるようになります。

プロパティ名	説明	プロパティ値
RDS_HOSTNAME	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
RDS_PORT	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。
RDS_DB_NAME	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
RDS_USERNAME	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。
RDS_PASSWORD	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

Elastic Beanstalk 環境と結合したデータベースインスタンスの設定の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

ドライバのダウンロード

`EntityFramework` を使用して開発環境用の `NuGet` パッケージとデータベースドライバをダウンロードしてインストールします。

.NET 用の共通エンティティフレームワークデータベースプロバイダ

- SQL Server – `Microsoft.EntityFrameworkCore.SqlServer`
- MySQL – `Pomelo.EntityFrameworkCore.MySql`
- PostgreSQL – `Npgsql.EntityFrameworkCore.PostgreSQL`

データベースへの接続

Elastic Beanstalk は、環境プロパティでアタッチされた DB インスタンスの接続情報を提供します。`ConfigurationManager.AppSettings` を使用してプロパティを読み取り、データベース接続を設定します。

Example Helpers.cs - 接続文字列メソッド

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Web;

namespace MVC5App.Models
{
  public class Helpers
  {
    public static string GetRDSConnectionString()
    {
      var appConfig = ConfigurationManager.AppSettings;

      string dbname = appConfig["RDS_DB_NAME"];

      if (string.IsNullOrEmpty(dbname)) return null;

      string username = appConfig["RDS_USERNAME"];
      string password = appConfig["RDS_PASSWORD"];
      string hostname = appConfig["RDS_HOSTNAME"];
      string port = appConfig["RDS_PORT"];

      return "Data Source=" + hostname + ";Initial Catalog=" + dbname + ";User ID=" +
 username + ";Password=" + password + ";";
    }
  }
}
```

接続文字列を使用してデータベースコンテキストを初期化します。

Example DBContext.cs

```
using System.Data.Entity;
using System.Security.Claims;
```

```
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace MVC5App.Models
{
  public class RDSContext : DbContext
  {
    public RDSContext()
      : base(GetRDSConnectionString())
    {
    }

    public static RDSContext Create()
    {
      return new RDSContext();
    }
  }
}
```

AWS Toolkit for Visual Studio

Visual Studio はさまざまなプログラミング言語とアプリケーションの種類用のテンプレートを提供します。いずれのテンプレートを使用して開始できます。 AWS Toolkit for Visual Studio には、アプリケーションの開発をブートストラップする 3 つのプロジェクトテンプレート、 AWS コンソールプロジェクト、 AWS ウェブプロジェクト、 AWS および空のプロジェクトも用意されています。この例では、新しい ASP.NET ウェブアプリケーションを作成します。

新しい ASP.NET ウェブアプリケーションプロジェクトを作成するには

1. Visual Studio の [File] メニューで [New] をクリックし、[Project] をクリックします。
2. [New Project] ダイアログボックスで [Installed Templates] をクリックし、[Visual C#] をクリックした後、[Web] をクリックします。[ASP.NET Empty Web Application] をクリックし、プロジェクト名を入力して [OK] をクリックします。

プロジェクトを実行するには

次のいずれかを行ってください。

1. F5 を押します。

2. [Debug] メニューから [Start Debugging] を選択します。

ローカルでテストします

Visual Studio では、簡単にローカルでアプリケーションをテストできます。ASP.NET ウェブアプリケーションをテストするか実行するには、ウェブブラウザが必要です。Visual Studio には、Internet Information Services（IIS）、IIS Express、組み込みの Visual Studio Development Server などのオプションが用意されています。各オプションの詳細や最適なオプションの決定方法については、「ASP.NET Web プロジェクト用の Visual Studio の Web サーバー」を参照してください。

Elastic Beanstalk 環境の作成

アプリケーションをテストすると、Elastic Beanstalk にデプロイする準備が完了します。

> **Note**
>
> 設定ファイルはアーカイブに含めるプロジェクトの一部である必要があります。または、プロジェクトに設定ファイルを含める代わりに、Visual Studio を使用してプロジェクトフォルダにすべてのファイルをデプロイできます。Solution Explorer でプロジェクト名を右クリックし、[Properties] をクリックします。[Package/Publish Web] タブをクリックします。[Items to deploy] セクションで、ドロップダウンリストの [All Files in the Project Folder] を選択します。

Toolkit AWS for Visual Studio を使用してアプリケーションを Elastic Beanstalk にデプロイするには

1. Solution Explorer で、アプリケーションを右クリックし、発行先 AWSを選択します。
2. [に発行 AWS] ウィザードで、アカウント情報を入力します。
   a. デプロイに使用するAWS アカウントで、アカウントを選択するか、[Other] (その他) を選択して新しいアカウント情報を入力します。
   b. [Region] で、アプリケーションをデプロイするリージョンを選択します。利用可能な AWS リージョンの詳細については、AWS Elastic Beanstalk 「」の「エンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。Elastic Beanstalk でサポートされていないリージョンを選択すると、Elastic Beanstalk にデプロイするオプションは利用できなくなります。
   c. [Deploy new application with template] をクリックし、[Elastic Beanstalk] を選択します。次に、[次へ] をクリックします。

3. [Application] ページで、アプリケーションの詳細を入力します。

   a. [Name] に、アプリケーション名を入力します。

   b. [Description] にアプリケーションの説明を入力します。この手順は省略可能です。

   c. アプリケーションのバージョンラベルは、[Deployment version label (デプロイバージョンラベル)] に自動的に表示されます。

   d. 変更したファイルのみデプロイするため、[Deploy application incrementally] を選択します。すべてのファイルではなく変更されたファイルのみ更新するため、増分デプロイの方が高速です。このオプションを選択した場合、アプリケーションバージョンは Git コミット ID から設定されます。アプリケーションの増分をデプロイしないを選択した場合、[Deployment version label] ボックスでバージョンラベル付けを更新できます。

   e. [Next] をクリックします。

4. [Environment] ページで、環境の詳細を説明します。

   a. [Create a new environment for this application] を選択します。

   b. [Name] に、環境の名前を入力します。

   c. [Description] で、環境の特性を説明します。この手順は省略可能です。

   d. 使用する環境の [Type] を選択します。

      [Load balanced, auto scaled] 環境または [Single instance] 環境のいずれかを選択できます。詳細については、「環境タイプ」を参照してください。

      > **Note**
      > 単一インスタンスの環境の場合は、負荷分散、Auto Scaling、ヘルスチェック URL の設定は適用されません。

   e. [Environment URL] ボックスにカーソルを移動すると、環境の URL が自動的に表示されます。

   f. [Check availability (稼働率をチェックする)] をクリックして、環境 URL が利用できることを確認します。

   g. [Next] をクリックします。

5. [AWS Options] ページで、デプロイの他のオプションとセキュリティ情報を設定します。

   a. [Container Type] で、[64bit Windows Server 2012 running IIS 8] または [64bit Windows Server 2008 running IIS 7.5] を選択します。

   b. [Instance Type] で [Micro] を選択します。

   c. [Key pair] で [Create new key pair] を選択します。新しいキーペアの名前 (この例では、**myuswestkeypair**) を入力して、[OK] をクリックします。キーペアを使用すると、Amazon EC2 インスタンスへのリモートデスクトップアクセスが可能になります。Amazon EC2 キーペアの詳細については、Amazon Elastic Compute Cloud ユーザーガイドの「Using Credentials」を参照してください。

   d. インスタンスプロファイルを選択します。

      インスタンスプロファイルがない場合は、[Create a default instance profile] を選択します。Elastic Beanstalk でのインスタンスプロファイルの使用については、「Elastic Beanstalk インスタンスプロファイルの管理」を参照してください。

   e. 環境で使用したいカスタム VPC がある場合は、[Launch into VPC] をクリックします。次のページで、VPC 情報を設定できます。Amazon VPC の詳細については、「Amazon Virtual Private Cloud (Amazon VPC)」を参照してください。Support されているレガシーではないコンテナタイプのリストについては、「the section called “一部のプラットフォームバージョンがレガシーとマークされているのはなぜですか?”」を参照してください。

   f. [Next] をクリックします。

6. VPC 内で環境を起動することを選択した場合は、[VPC Options] ページが表示されます。そうでない場合は、[Additional Options] ページが表示されます。ここでは、VPC のオプションを設定します。

   a. 環境を起動する VPC の VPC ID を選択します。

   b. 負荷分散されたスケーラブルな環境では、Elastic Load Balancing をインターネットで利用できないようにする場合、[ELB Scheme] で [プライベート] を選択します。

      単一インスタンスの環境の場合、環境にロードバランサーがないため、このオプションは適用されません。詳細については、「」を参照してください環境タイプ

   c. 負荷分散されたスケーラブルな環境では、Elastic Load Balancing と EC2 インスタンスのサブネットを選択します。パブリックサブネットとプライベートサブネットを作成した場

合、Elastic Load Balancing と EC2 インスタンスが正しいサブネットと関連付けられていることを確認してください。Amazon VPC のデフォルトでは、10.0.0.0/24 を使用するデフォルトパブリックサブネットと 10.0.1.0/24 を使用するプライベートサブネットを作成します。既存のサブネットは https://console.aws.amazon.com/vpc/ の Amazon VPC コンソールで表示できます。

単一インスタンスの環境の場合、VPC に必要なのは、インスタンスのパブリックサブネットのみです。この環境には、ロードバランサーがないため、ロードバランサーのサブネット選択は適用されません。詳細については、「」を参照してください環境タイプ

d. 負荷分散されたスケーラブルな環境の場合は、必要に応じてインスタンス用に作成したセキュリティグループを選択します。

   単一インスタンスの環境では、NAT デバイスは必要ありません。default セキュリティグループを選択します。Elastic Beanstalk は、インスタンスがインターネットにアクセスできるように、Elastic IP アドレスをインスタンスに割り当てます。

e. [Next] をクリックします。

7. [Application Options] ページで、アプリケーションのオプションを設定します。

   a. [Target framework] として、[.NET Framework 4.0] を選択します。

   b. Elastic Load Balancing では、ヘルスチェックを使用して、アプリケーションを実行している Amazon EC2 インスタンスが正常であるかどうかを確認します。ヘルスチェックでは、指定された URL を設定間隔で調査して、インスタンスのヘルスステータスを確認します。デフォルト URL をオーバーライドしてアプリケーションの既存のリソース (たとえば、`/myapp/index.aspx`) に一致させるには、[Application health check URL] ボックスに URL を入力します。アプリケーションのヘルスチェックの詳細については、「ヘルスチェック」を参照してください。

   c. アプリケーションに影響する重要なイベントに関する Amazon Simple Notification Service（Amazon SNS）通知を受信する場合は、E メールアドレスを入力します。

   d. [Application Environment] セクションでは、アプリケーションを実行している Amazon EC2 インスタンスの環境変数を指定できます。この設定では、環境間を移動するときにソースコードの再コンパイルを不要にすることで、性が向上します。

   e. アプリケーションのデプロイに使用するアプリケーション認証情報オプションを選択します。

   f. [Next] をクリックします。

8. 以前に Amazon RDS データベースをセットアップしたことがある場合には、[Amazon RDS DB Security Group] ページが表示されます。Elastic Beanstalk 環境を Amazon RDS DB インスタンスに接続する場合、1 つ以上のセキュリティグループを選択します。それ以外の場合、次のステップに進みます。準備が完了したら、[Next] をクリックします。

9. デプロイのオプションを確認します。すべて問題なければ、[デプロイ] をクリックします。

   ASP.NET プロジェクトはウェブデプロイファイルとしてエクスポートされ、Amazon S3 にアップロードされ、Elastic Beanstalk に新しいアプリケーションバージョンとして登録されます。Elastic Beanstalk デプロイ機能は、新しくデプロイされたコードで利用できるようになるまで、環境を監視します。[env:<environment name>] タブに、環境のステータスが表示されます。

環境を終了する

未使用の AWS リソースに対して料金が発生しないように、 AWS Toolkit for Visual Studio を使用して実行中の環境を終了できます。

> **Note**
> いつでも、また同じバージョンを使用して新しい環境を起動できます。

環境を終了するには

1. [AWS Explorer] でElastic Beanstalk ノードとアプリケーションノードを展開します。アプリケーション環境を右クリックして、[環境の終了] を選択します。
2. プロンプトが表示されたら、[はい] をクリックして、環境を終了することを確認します。Elastic Beanstalk が環境で実行されている AWS リソースを終了するまでに数分かかります。

   > **Note**
   > 環境を終了すると、終了した環境に関連付けられていた CNAME はすべてのユーザーが使用できるようになります。

環境へのデプロイ

アプリケーションをテストしたので、アプリケーションを編集して再デプロイしすぐに結果を表示することは簡単です。

ASP.NET ウェブアプリケーションを編集して再デプロイするには

1. [Solution Explorer] でアプリケーションを右クリックし、[Republish to Environment <***your ## name***>] をクリックします。[Re-publish to AWS Elastic Beanstalk] (Amazon Elastic Beanstalk への再発行) ウィザードが開きます。

2. デプロイの詳細を確認し、[デプロイ] をクリックします。

   > **Note**
   >
   > 設定を変更する場合、[キャンセル] をクリックし、代わりに [への発行 AWS] ウィザードを使用します。手順については、「Elastic Beanstalk 環境の作成」を参照してください。

   更新した ASP.NET ウェブプロジェクトは新しいバージョンラベルが付いたウェブデプロイファイルとしてエクスポートされ、Amazon S3 にアップロードされ、Elastic Beanstalk に新しいアプリケーションバージョンとして登録されます。Elastic Beanstalk デプロイ機能は、新しくデプロイされたコードで利用できるようになるまで、既存の環境を監視します。[env:<***environment name***>] タブに環境のステータスが表示されます。

以前のアプリケーションバージョンにロールバックする必要がある場合などは、既存のアプリケーションを既存の環境にデプロイすることもできます。

既存のアプリケーションバージョンを既存の環境にデプロイするには

1. [AWS Explorer] で Elastic Beanstalk ノードを展開して、Elastic Beanstalk アプリケーションを右クリックします。[View Status (ステータスの表示)] を選択します。
2. [App: <***application name***>] タブで [バージョン] をクリックします。

3. デプロイするアプリケーションバージョンをクリックし、[Publish Version (バージョンの発行)] をクリックします。

4. [Publish Application Version (アプリケーションバージョンの発行)] ウィザードで、[次へ] をクリックします。

5. デプロイのオプションを確認し、[デプロイ] をクリックします。

   ASP.NET プロジェクトはウェブデプロイファイルとしてエクスポートされ、Amazon S3 にアップロードされます。Elastic Beanstalk デプロイ機能は、新しくデプロイされたコードで利用できるようになるまで、環境を監視します。[env:<*environment name*>] タブに、環境のステータスが表示されます。

Elastic Beanstalk アプリケーション環境の管理

AWS Toolkit for Visual Studio と AWS マネジメントコンソールを使用すると、アプリケーション環境で使用される AWS リソースのプロビジョニングと設定を変更できます。 AWS マネジメントコンソールを使用してアプリケーション環境を管理する方法については、「」を参照してください[Elastic Beanstalk での環境の作成](#)。ここでは、アプリケーション環境設定の一部として AWS Toolkit for Visual Studio で編集できる特定のサービス設定について説明します。

環境設定を変更する

アプリケーションをデプロイすると、Elastic Beanstalk は多数の AWS クラウドコンピューティングサービスを設定します。 AWS Toolkit for Visual Studio を使用して、これらの個々のサービスの設定方法を制御できます。

アプリケーションの環境設定を

- Elastic Beanstalk ノードとアプリケーションノードを展開します。次に、[AWS Explorer] で Elastic Beanstalk 環境を右クリックします。[View Status (ステータスの表示)] を選択します。

  次の設定を編集できます。

  - サーバー
  - 負荷分散
  - Auto Scaling
  - 通知
  - 環境プロパティ

AWS Toolkit for Visual Studio を使用した EC2 サーバーインスタンスの設定

Amazon Elastic Compute Cloud （Amazon EC2）は、Amazon のデータセンターにあるサーバーインスタンスの起動と管理に使用するウェブサービスです。必要に応じて、任意の合法的な用途で Amazon EC2 サーバーインスタンスをいつでも使用できます。インスタンスはさまざまなサイズや設定で使用できます。詳細については、Amazon EC2 を参照してください。

Toolkit AWS for Visual Studio のアプリケーション環境タブ内の Server タブを使用して、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を編集できます。

Amazon EC2 インスタンスタイプ

[インスタンスタイプ] には、Elastic Beanstalk アプリケーションで使用できるインスタンスタイプが表示されます。インスタンスタイプを変更して、アプリケーションに最適な特性 (メモリサイズや CPU 能力など) を持つサーバーを選択します。例えば、高負荷の操作を長時間実行するアプリケーションでは、より大きい CPU やメモリが必要となる場合があります。

Elastic Beanstalk アプリケーションで使用可能な Amazon EC2 インスタンスタイプの詳細については、Amazon Elastic Compute Cloud ユーザーガイドの「インスタンスタイプ」を参照してください。

Amazon EC2 セキュリティグループ

Amazon EC2 セキュリティグループを使用すると、Elastic Beanstalk アプリケーションへのアクセスを制御できます。セキュリティグループとは、インスタンスのファイアウォールのルールを定義するものです。このルールでは、どの着信ネットワークトラフィックをご使用のインスタンスに配信するかを指定します。他の着信トラフィックはすべて破棄されます。グループのルールはいつでも変更することができます。実行中のすべてのインスタンスと、今後起動されるインスタンスについて、新しいルールが自動的に実施されます。

Amazon EC2 セキュリティグループは、 AWS マネジメントコンソールまたは AWS Toolkit for Visual Studio を使用して設定できます。1 つまたは複数の Amazon EC2 セキュリティグループ名の名前を [EC2 Security Groups] テキストボックスに入力して、Elastic Beanstalk アプリケーションへのアクセスを制御する Amazon EC2 セキュリティグループを指定できます (グループ間はコンマで区切ります)。

> **Note**
>
> アプリケーションのヘルスチェックを有効にする場合、ソース CIDR 範囲として 0.0.0.0/0 からポート 80 （HTTP）がアクセスできることを確認してください。ヘルスチェックの詳細については、「ヘルスチェック」を参照してください。

AWS Toolkit for Visual Studio を使用してセキュリティグループを作成するには

1. Visual Studio の [AWS Explorer] で [Amazon EC2] ノードを展開し、[Security Groups] (セキュリティグループセキュリティグループ) をダブルクリックします。
2. [Create Security Group] をクリックし、セキュリティグループの名前と説明を入力します。
3. [OK] をクリックします。

Amazon EC2 セキュリティグループの詳細については、Amazon Elastic Compute Cloud ユーザーガイドの「セキュリティグループの使用」を参照してください。

Amazon EC2 のキーペア

Elastic Beanstalk アプリケーション用にプロビジョニングされた Amazon EC2 インスタンスには、Amazon EC2 のキーペアを使用して安全にログインできます。

> **Important**
>
> Elastic Beanstalk 提供の Amazon EC2 インスタンスにアクセスする前に、Amazon EC2 のキーペアを作成し、Elastic Beanstalk 提供の Amazon EC2 インスタンスが Amazon EC2 のキーペアを使用するよう設定する必要があります。アプリケーションを Elastic Beanstalk にデプロイするときに、 AWS Toolkit for Visual Studio 内の Publish to AWS ウィザードを使用してキーペアを作成できます。Toolkit を使用してさらにキーペアを作成する場合は、次の手順に従ってください。または、AWS マネジメントコンソールを使用して、Amazon EC2 のキーペアを設定することもできます。Amazon EC2 のキーペアを作成する手順については、Amazon Elastic Compute Cloud 入門ガイドを参照してください。

[Existing Key Pair] テキストボックスを使用すると、Elastic Beanstalk アプリケーションを実行している Amazon EC2 インスタンスに安全にログインするために使用できる Amazon EC2 のキーペアの名前を指定できます。

Amazon EC2 のキーペアの名前を指定するには

1. [Amazon EC2] ノードを展開し、[Key Pairs] をダブルクリックします。
2. [Create Key Pair] をクリックし、キーペア名を入力します。
3. [OK] をクリックします。

Amazon EC2 のキーペアの詳細については、Amazon Elastic Compute Cloud ユーザーガイドの「Amazon EC2 認証情報の使用」のページを参照してください。Amazon EC2 インスタンスへの接続の詳細については、「サーバーインスタンスの一覧表示と接続」を参照してください。

間隔のモニタリング

デフォルトでは、基本的な Amazon CloudWatch メトリクスだけが有効化されています。5 分周期でデータを返します。 AWS Toolkit for Eclipseの環境の [Configuration] (設定) タブの [Server] (サーバー) セクションで、[Monitoring Interval] (モニタリング間隔) で [1 minute] (1 分)を選択すると、より詳細な 1 分間隔の CloudWatch メトリクスを有効化することもできます。

> **Note**
>
> Amazon CloudWatch の利用料金で 1 分間隔のメトリクスに申し込むことができます。詳細については、Amazon CloudWatch を参照してください。

カスタム AMI ID

AWS Toolkit for Eclipseの環境で、[Configuration] (設定) タブの [Server] (サーバー) セクションの [Custom AMI ID] (カスタム AMI ID) ボックスにカスタム AMI の ID を入力して、Amazon EC2 インスタンスで使用するデフォルトの AMI を独自のカスタム AMI に置き換えることができます。

> **Important**
>
> 独自の AMI の使用は高度な作業であるため、注意が必要です。カスタム AMI が必要な場合は、デフォルトの Elastic Beanstalk AMI を変更して使用することをお勧めします。正常と見なされるには、Amazon EC2 インスタンスが、ホストマネージャの実行を含む Elastic Beanstalk の一連の要件を満たす必要があります。これらの要件を満たさないと、環境が正常に動作しない可能性があります。

AWS ツールキット for Visual Studio を使用した Elastic Load Balancing の設定

Elastic Load Balancing は、アプリケーションの可用性とスケーラビリティの向上に役立つ Amazon のウェブサービスです。このサービスによって、アプリケーションの負荷を簡単に複数の Amazon EC2 インスタンスに分散できます。Elastic Load Balancing による冗長化で可用性が改善され、アプリケーションのトラフィック増加に対応できます。

Elastic Load Balancing を使用すると、実行しているすべてのインスタンス間で、アプリケーションの着信トラフィックを配信して負荷分散を行うことができます。また、アプリケーションの処理能力を増やす必要があるときには、新しいインスタンスを簡単に追加することもできます。

アプリケーションをデプロイすると、Elastic Beanstalk によって自動的に Elastic Load Balancing がプロビジョニングされます。 AWS Toolkit for Visual Studio のアプリケーション環境タブ内の Load Balancer タブを使用して、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を編集できます。

ここでは、アプリケーションで設定できる Elastic Load Balancing パラメータについて説明します。

ポート

Elastic Beanstalk アプリケーションへのリクエストを処理するためにプロビジョニングされたロードバランサーは、アプリケーションを実行している Amazon EC2 インスタンスにリクエストを送信します。プロビジョニングされたロードバランサーは、HTTP ポートと HTTPS ポートでリクエストをリッスンし、 AWS Elastic Beanstalk アプリケーション内の Amazon EC2 インスタンスにリクエストをルーティングできます。デフォルトでは、ロードバランサーは HTTP ポートのリクエストを処理します。少なくともいずれかのポート (HTTP または HTTPS) を有効にする必要があります。

> **Important**
>
> 指定したポートがロックされていないことを確認してください。ロックされている場合、ユーザーは Elastic Beanstalk アプリケーションに接続できません。

HTTP ポートを制御する

HTTP ポートをオフにするには、[HTTP Listener Port] で [OFF] を選択します。HTTP ポートを有効にするには、リストから HTTP ポート ([80] など) を選択します。

> **Note**
>
> デフォルトポート 80 以外のポート (例: ポート 8080) を使用して環境にアクセスする場合は、既存のロードバランサーにリスナーを追加し、そのポートでリッスンするようにリスナーを設定します。
> 例えば、Classic Load Balancer 用のAWS CLI を使用して、次のコマンドを入力します。*LOAD_BALANCER_NAME* は Elastic Beanstalk のロードバランサーの名前に置き換えてください。
>
> ```
> aws elb create-load-balancer-listeners --load-balancer-name LOAD_BALANCER_NAME
> --listeners "Protocol=HTTP, LoadBalancerPort=8080, InstanceProtocol=HTTP,
> InstancePort=80"
> ```
>
> 例えば、Application Load Balancer 用のAWS CLI を使用して、次のコマンドを入力します。*LOAD_BALANCER_ARN* は Elastic Beanstalk のロードバランサーの ARN に置き換えてください。
>
> ```
> aws elbv2 create-listener --load-balancer-arn LOAD_BALANCER_ARN --protocol HTTP
> --port 8080
> ```
>
> Elastic Beanstalk を使用して環境を監視する場合は、ポート 80 のリスナーを削除しないでください。

HTTPS ポートを制御する

Elastic Load Balancing は、ロードバランサーへのクライアント接続のトラフィックを暗号化するために、HTTPS/TLS プロトコルをサポートしています。ロードバランサーから EC2 インスタンスへの接続では、プレーンテキストの暗号化が使用されます。デフォルトで、HTTPS ポートは無効です。

HTTPS ポートを有効にするには

1. AWS Certificate Manager (ACM) を使用して新しい証明書を作成するか、証明書とキーを AWS Identity and Access Management (IAM) にアップロードします。ACM 証明書のリクエストの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のリクエスト」を参照してください。ACM へのサードパーティー証明書のインポートの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のインポート」を参照してください。ACM がお客様のリー

ジョンで利用できない場合は、(IAM) を使用してサードパーティー証明書をアップロードします。 AWS Identity and Access Management ACM および IAM サービスは証明書を保存し、SSL 証明書の Amazon リソースネーム (ARN) を提供します。証明書の作成と IAM へのアップロードに関する詳細については、IAM ユーザーガイドの「サーバー証明書の使用」を参照してください。

2. [HTTPS Listener Port (HTTPS リスナーポート)] のポートを選択して、HTTPS ポートを指定します。

3. [SSL 証明書 ID] に、SSL 証明書の Amazon リソースネーム (ARN) を入力します。例えば、**`arn:aws:iam::123456789012:server-certificate/abc/certs/build`**、**`arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`** などです。ステップ 1 で作成またはアップロードした SSL 証明書を使用します。

HTTPS ポートをオフにするには、[HTTPS Listener Port] で [OFF] を選択します。

ヘルスチェック

ヘルスチェックの定義には、インスタンスのヘルスを照会する URL が含まれます。デフォルトでは、Elastic Beanstalk はレガシーではないコンテナの場合は TCP:80 を使用し、レガシーコンテナの場合は HTTP:80 を使用します。デフォルト URL をオーバーライドしてアプリケーションの既存のリソース (たとえば、`/myapp/default.aspx`) に一致させるには、[Application Health Check URL (アプリケーションヘルスチェック URL)] ボックスに URL を入力します。デフォルトの URL をオーバーライドすると、Elastic Beanstalk は HTTP を使用してリソースを照会します。レガシーコンテナタイプを使用しているかどうかを確認するには、「the section called “一部のプラットフォームバージョンがレガシーとマークされているのはなぜですか?”」を参照してください。

[Load Balancing] パネルの [EC2 Instance Health Check] を使用して、ヘルスチェックの設定を制御できます。

ヘルスチェックの定義には、インスタンスのヘルスを照会する URL が含まれます。[Application Health Check URL] ボックスに入力することによって、デフォルト URL をオーバーライドし、アプリケーションの既存のリソース (たとえば、`/myapp/index.jsp`) に一致させます。

次の一覧では、アプリケーションで設定できるヘルスチェックパラメータについて説明します。

- [Health Check Interval (seconds)] には、Elastic Load Balancing がアプリケーションの Amazon EC2 インスタンスの各ヘルスチェックを待機する秒数を入力します。
- [Health Check Timeout (seconds)] には、Elastic Load Balancing がインスタンスの応答がないとみなす応答待機時間の秒数を入力します。
- [Healthy Check Count Threshold] および [Unhealthy Check Count Threshold] には、Elastic Load Balancing がインスタンスのヘルスステータスを変更するまでの URL 探索の連続成功回数または失敗回数を指定します。たとえば、[ヘルスチェック失敗数のしきい値] ボックスに **5** と指定した場合、Elastic Load Balancing がヘルスチェックを失敗とみなすには、URL がエラーメッセージまたはタイムアウトを 5 回連続して返す必要があります。

セッション

デフォルトでは、ロードバランサーは負荷が最小になるように、各リクエストを個別にサーバーインスタンスにルーティングします。比較すると、スティッキーセッションの場合、セッション中にユーザーから受信するすべてのリクエストが、同じサーバーインスタンスに送信されるように、ユーザーのセッションを特定のサーバーインスタンスにバインドします。

アプリケーションでスティッキーセッションが有効な場合、Elastic Beanstalk は、ロードバランサーで生成された HTTP Cookie を使用します。ロードバランサーは、ロードバランサーが生成する特別な Cookie を使って、各リクエストのアプリケーションインスタンスを追跡します。ロードバランサーがリクエストを受け取ると、まずこの Cookie がリクエスト内にあるかどうかを調べます。ある場合は、Cookie で指定されたアプリケーションインスタンスにリクエストが送信されます。Cookie がない場合、ロードバランサーは、既存の負荷分散アルゴリズムに基づいてアプリケーションインスタンスを選択します。同じユーザーからの以降のリクエストをそのアプリケーションインスタンスにバインドするため、応答に Cookie が挿入されます。ポリシー設定では、各 Cookie の有効期間を設定する Cookie 期限を定義します。

[Load Balancer] タブの [Sessions] セクションを使用して、アプリケーションのロードバランサーでスティッキーセッションを使用できるようにするかどうかを指定できます。

Elastic Load Balancing の詳細については、Elastic Load Balancing デベロッパーガイドを参照してください。

Toolkit AWS for Visual Studio を使用した Auto Scaling の設定

Amazon EC2 Auto Scaling は、ユーザーが定義したトリガーに基づいて、Amazon EC2 インスタンスを自動的に起動または終了するように設計された Amazon のウェブサービスです。ユーザーは

Auto Scaling グループをセットアップし、そのグループにトリガーを関連付けることで、帯域幅の使用や CPU の使用率などのメトリクスに基づいて、コンピューティングリソースを自動的にスケールできます。Amazon EC2 Auto Scaling は Amazon CloudWatch と連携して、アプリケーションを実行するサーバーインスタンスのメトリクスを取得します。

Amazon EC2 Auto Scaling によって、Amazon EC2 インスタンスのグループを利用して、自動的に数を増減できるようにさまざまなパラメータを設定できます。Amazon EC2 Auto Scaling は、アプリケーションのトラフィックの変化をシームレスに処理できるように、Amazon EC2 インスタンスのグループを追加または削除できます。

Amazon EC2 Auto Scaling は、起動した各 Amazon EC2 インスタンスの状態もモニタリングします。インスタンスが予期せずに終了した場合、Amazon EC2 Auto Scaling は終了を検出し、代わりのインスタンスを起動します。この機能を使用すると、任意の固定の Amazon EC2 インスタンス数を自動的に維持できます。

Elastic Beanstalk はアプリケーション用に Amazon EC2 Auto Scaling のプロビジョニングを行います。 AWS Toolkit for Visual Studio のアプリケーション環境タブ内の Auto Scaling タブを使用して、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を編集できます。

ここでは、アプリケーションの Auto Scaling パラメータの設定方法について説明します。

設定の起動

起動設定を編集すると、Elastic Beanstalk アプリケーションによる Amazon EC2 Auto Scaling リソースのプロビジョニング方法を制御できます。

[Minimum Instance Count (最小インスタンス数)] ボックスと [Maximum Instance Count (最大インスタンス数)] ボックスを使用して、Elastic Beanstalk アプリケーションが使用する Auto Scaling グループの最小サイズと最大サイズを指定できます。

> **Note**
>
> 固定の Amazon EC2 インスタンス数を維持するには、[Minimum Instance Count] と [Maximum Instance Count] を同じ値に設定します。

[Availability Zones] ボックスでは、Amazon EC2 インスタンスを維持するアベイラビリティーゾーンの数を指定できます。フォールトトレラントアプリケーションを構築する場合、この数を設定するこ

とをお勧めします。1 つのアベイラビリティーゾーンが停止しても、インスタンスは他のアベイラビリティーゾーンで実行されます。

> **Note**
>
> 現在、インスタンスを維持するアベイラビリティーゾーンを指定することはできません。

トリガー

トリガーとは、インスタンス数を増やす (スケールアウト) タイミングや、インスタンス数を減らす (スケールイン) タイミングをシステムに指示するために設定できる Amazon EC2 Auto Scaling のメカニズムです。CPU の使用率など、Amazon CloudWatch に発行された任意のメトリクスについて、トリガーが発生するよう設定し、指定した条件を満たしているかどうかを判断することができます。メトリクスについて指定した条件の上限または下限を、指定した期間超過すると、トリガーによって Scaling Activity という長時間実行されるプロセスが起動されます。

AWS Toolkit for Visual Studio を使用して、Elastic Beanstalk アプリケーションのスケーリングトリガーを定義できます。

Amazon EC2 Auto Scaling のトリガーは、インスタンス固有の Amazon CloudWatch メトリクスを監視して動作します。トリガーには、CPU 使用率、ネットワークトラフィック、ディスクアクティビティが含まれます。[Trigger Measurement] 設定を使用して、トリガーのメトリクスを選択します。

次のリストでは、 AWS マネジメントコンソールを使用して設定できるトリガーパラメータについて説明します。

- トリガーに使用する統計を指定できます。[Trigger Statistic] に対して選択できるのは、[Minimum]、[Maximum]、[Sum]、または [Average] です。
- [Unit of Measurement] には、トリガー測定の単位を指定します。
- [Measurement Period] ボックスの値は、Amazon CloudWatch がトリガーのメトリクスを計測する頻度を指定します。[Breach Duration] は、トリガーが発生するまでに、定義した限度 ([Upper Threshold] と [Lower Threshold] に指定した値) をメトリックが超過できる時間を示します。
- [上限超過スケール増分] と [下限超過スケール増分] には、規模の拡大や縮小を実行する際に追加または削除する Amazon EC2 インスタンスの数を指定します。

Amazon EC2 Auto Scaling の詳細については、Amazon Elastic Compute Cloud ドキュメントの「Amazon EC2 Auto Scaling」セクションを参照してください。

AWS Toolkit for Visual Studio を使用した通知の設定

Elastic Beanstalk では、Amazon Simple Notification Service (Amazon SNS) を使用して、アプリケーションに影響を与える重要なイベントについて通知します。Amazon SNS 通知を有効化するには、[Email Address] ボックスに電子メールアドレスを入力します。Amazon SNS 通知を無効にするには、ボックスから電子メールアドレスを削除します。

AWS Toolkit for Visual Studio を使用した .NET コンテナの設定

[Container/.NET Options] パネルでは、Amazon EC2 インスタンスの動作を微調整し、Amazon S3 のログの更新を有効または無効にすることができます。 AWS Toolkit for Visual Studio を使用して、コンテナ情報を設定できます。

> **Note**
>
> 環境の CNAME を切り替えることで、ダウンタイムなしで設定を変更できます。詳細については、「Elastic Beanstalk を使用したブルー/グリーンデプロイ」を参照してください。

必要に応じて、パラメータの数を拡張できます。パラメータの拡張の詳細については、「オプション設定」を参照してください。

Elastic Beanstalk アプリケーションの [Container/.NET] パネルにアクセスするには

1. AWS Toolkit for Visual Studio で、Elastic Beanstalk ノードとアプリケーションノードを展開します。
2. 次に、[AWS Explorer] で Elastic Beanstalk 環境をダブルクリックします。
3. [Overview] ペインの下部にある [Configuration] タブをクリックします。
4. [Container] で、コンテナのオプションを設定できます。

.NET コンテナのオプション

アプリケーションの .NET Framework のバージョンを選択できます。[Target runtime] として、2.0 または 4.0 を選択します。32 ビットアプリケーションを有効にするには、[Enable 32-bit Applications] を選択します。

アプリケーションの設定

[Application Settings] セクションで、アプリケーションコードから読み取ることのできる環境変数を指定できます。

アカウントの管理

テスト、ステージング、本番稼働など、さまざまなタスクを実行するように異なる AWS アカウントを設定する場合は、 AWS Toolkit for Visual Studio を使用してアカウントを追加、編集、削除できます。

複数のアカウントを管理するには

1. Visual Studio の [View] (ビュー) メニューで、[AWS Explorer] をクリックします。
2. [Account] リストの横にある [Add Account] ボタンをクリックします。

   [Add Account] ダイアログボックスが表示されます。

3. 必要な情報を入力します。
4. [AWS Explorer] タブにアカウント情報が表示されます。Elastic Beanstalk に公開する場合、使用するアカウントを選択できます。

サーバーインスタンスの一覧表示と接続

Toolkit AWS for Visual Studio または AWS マネジメントコンソールを使用して、Elastic Beanstalk アプリケーション環境を実行している Amazon EC2 インスタンスのリストを表示できます。これらのインスタンスには、リモートデスクトップ接続を使用して接続できます。 AWS マネジメントコンソールを使用してサーバーインスタンスを一覧表示して接続する方法については、「」を参照してくださいサーバーインスタンスの一覧表示と接続。次のセクションでは、 AWS Toolkit for Visual Studio を使用してサーバーインスタンスを表示および接続する手順を示します。

環境の Amazon EC2 インスタンスを表示して接続するには

1. Visual Studio の [AWS Explorer] で [Amazon EC2] ノードを展開し、[Instances] (インスタンス) をダブルクリックします。
2. アプリケーションのロードバランサーで実行されている Amazon EC2 インスタンスのインスタンス ID を [Instance (インスタンス)] で右クリックし、コンテキストメニューから [Open Remote Desktop (オープンリモートデスクトップ)] を選択します。
3. [Use EC2 keypair to log on] を選択し、アプリケーションのデプロイに使用したプライベートキーファイルの内容を [Private key] ボックスに貼り付けます。または、[User name] テキストボックスと [Password] テキストボックスにユーザー名とパスワードを入力します。

   > **Note**
   >
   > キーペアが Toolkit 内に保存されている場合、テキストボックスは表示されません。

4. [OK] をクリックします。

アプリケーションの状態をモニタリングする

本番ウェブサイトを実行する場合、アプリケーションが利用可能であり、リクエストに応答するか確認することが重要です。アプリケーションの応答性のモニタリングを支援するために、Elastic Beanstalk はアプリケーションに関する統計情報を監視し、しきい値を超過するとトリガーされるアラートを作成する機能を提供しています。

Elastic Beanstalk で提供される状態モニタリングの詳細については、「ベーシックヘルスレポート」を参照してください。

Toolkit for Visual Studio または AWS マネジメントコンソールを使用して、 AWS アプリケーションに関する運用情報にアクセスできます。

Toolkit の [ステータス] フィールドでは環境のステータスとアプリケーションの状態が一目でわかります。

アプリケーションの状態を監視するには

1. AWS Toolkit for Visual Studio の AWS Explorer で、Elastic Beanstalk ノードを展開し、アプリケーションノードを展開します。

2. Elastic Beanstalk 環境を右クリックし、[View Status (ステータスの表示)] をクリックします。
3. アプリケーション環境タブで、[モニタリング] をクリックします。

   [モニタリング] パネルには、特定のアプリケーション環境に対するリソースの使用状況を示す一連のグラフが表示されます。

> **Note**
>
> デフォルトでは、時間範囲は 1 時間前に設定されます。この設定を変更するには、[時間範囲] リストで異なる時間範囲をクリックします。

AWS Toolkit for Visual Studio または AWS マネジメントコンソールを使用して、アプリケーションに関連付けられたイベントを表示できます。

アプリケーションイベントを表示するには

1. AWS Toolkit for Visual Studio の AWS Explorer で、Elastic Beanstalk ノードとアプリケーションノードを展開します。
2. [AWS Explorer] で Elastic Beanstalk 環境を右クリックし、[View Status] (ステータスの表示) をクリックします。
3. アプリケーション環境タブで [イベント] をクリックします。

デプロイメントツールを使用した Elastic Beanstalk アプリケーションの.NET でのデプロイ

AWS Toolkit for Visual Studio には、 AWS Toolkit のデプロイウィザードと同じ機能を提供するコマンドラインツールであるデプロイツールが含まれています。ビルドパイプラインまたはその他のスクリプトでデプロイツールを使用して、Elastic Beanstalk へのデプロイを自動化できます。

デプロイツールは、初期デプロイと再デプロイの両方に使用できます。デプロイツールを使用してアプリケーションをデプロイした後で、Visual Studio のデプロイウィザードを使用して再デプロイすることもできます。同様に、ウィザードを使用してデプロイした場合に、デプロイツールを使用して再デプロイすることもできます。

> **Note**
>
> デプロイツールでは、コンソールや EB CLI などの設定オプションの推奨値は適用されません。そのため、設定ファイルを使用して、環境を起動する際に必要な項目がすべて設定されていることを確認してください。

この章では、サンプル .NET アプリケーションを、まずデプロイツールを使用して Elastic Beanstalk にデプロイし、その後増分デプロイを使用して再デプロイする方法を順番に説明します。パラメータオプションを含め、デプロイツールに関する詳細な説明については、「デプロイツール」を参照してください。

前提条件

デプロイツールを使用するには、 AWS Toolkit for Visual Studio をインストールする必要があります。前提条件の詳細とインストール手順については、「AWS Toolkit for Microsoft Visual Studio」を参照してください。

デプロイツールは通常、Windows の以下のディレクトリのいずれかにインストールします。

32 ビット	64 ビット
`C:\Program Files\AWS Tools\Deployment Tool\awsdeploy.exe`	`C:\Program Files (x86)\AWS Tools\Deployment Tool\awsdeploy.exe`

Elastic Beanstalk にデプロイする

デプロイツールを使ってサンプルアプリケーションを Elastic Beanstalk にデプロイするには、まず `Samples` ディレクトリにある `ElasticBeanstalkDeploymentSample.txt` 構成ファイルを編集します。この設定ファイルには、アプリケーション名、アプリケーションバージョン、環境名、AWS アクセス認証情報など、アプリケーションをデプロイするために必要な情報が含まれています。構成ファイルを編集した後、コマンドラインを使用して、サンプルアプリケーションをデプロイします。ウェブデプロイファイルが Amazon S3 にアップロードされ、Elastic Beanstalk を使用した新しいアプリケーションバージョンとして登録されます。アプリケーションのデプロイには数分かかります。環境が正常であれば、デプロイツールが実行アプリケーション用に URL を出力します。

.NET アプリケーションを Elastic Beanstalk にデプロイする

1. デプロイツールがインストールされている Samples サブディレクトリから、 を開き、次の例のように AWS アクセスキーと AWS シークレットキーElasticBeanstalkDeploymentSample.txtを入力します。

```
### AWS Access Key and Secret Key used to create and deploy the application
 instance
AWSAccessKey = AKIAIOSFODNN7EXAMPLE
AWSSecretKey = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

> **Note**
>
> API アクセスには、アクセスキー ID とシークレットアクセスキーが必要です。 AWS アカウントのルートユーザー のアクセスキーの代わりに IAM ユーザーアクセスキーを使用します。アクセスキーの作成の詳細については、「IAM ユーザーガイド」の「IAM ユーザーのアクセスキーを管理する」を参照してください。

2. コマンドラインプロンプトで、以下のように入力します。

```
C:\Program Files (x86)\AWS Tools\Deployment Tool>awsdeploy.exe /w Samples
\ElasticBeanstalkDeploymentSample.txt
```

アプリケーションのデプロイには数分かかります。デプロイが成功すると、Application deployment completed; environment health is Green というメッセージが表示されます。

> **Note**
>
> 次のエラーが表示された場合は、CNAME がすでに存在します。
>
> ```
> [Error]: Deployment to AWS Elastic Beanstalk failed with exception: DNS name
> (MyAppEnv.elasticbeanstalk.com) is not available.
> ```
>
> CNAME は一意でなければならないため、Environment.CNAME の ElasticBeanstalkDeploymentSample.txt を変更する必要があります。

3. ウェブブラウザ内で、実行アプリケーションの URL に移動します。URL は <CNAME.elasticbeanstalk.com> という形式です (たとえば、**MyAppEnv.elasticbeanstalk.com**)。

オンプレミス .NET アプリケーションの Elastic Beanstalk への移行

AWS Elastic Beanstalk は、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を介してインターネットインフォメーションサービス (IIS) で実行されている Windows アプリケーションの効率的な移行パスを提供します。eb migrate コマンドは、IIS サイト、アプリケーション、仮想ディレクトリを自動的に検出し、設定を保持して、 AWS クラウドにデプロイします。

この組み込みの移行機能は、クラウド移行に通常関連する複雑さと時間を短縮するシンプルなアプローチを提供します。移行プロセスは、 への移行中にアプリケーションの機能と設定の整合性を維持するのに役立ちます AWS。

IIS アプリケーションの への移行に関する詳細な手順については AWS Elastic Beanstalk、このガイドのIIS アプリケーションの移行章を参照してください。

Elastic Beanstalk での Windows Server の廃止されたコンポーネントに関する推奨事項

このトピックでは、廃止された Windows Server 2012 R2 プラットフォームブランチでアプリケーションが現在実行されている場合の推奨事項を示します。また、 AWS サービス API エンドポイントと影響を受けるプラットフォームブランチでの TLS 1.0 および 1.1 プロトコルバージョンの非推奨サポートについても説明します。

Windows Server 2012 R2 プラットフォームブランチの廃止

Elastic Beanstalk は、2023 年 12 月 4 日に Windows Server 2012 R2 プラットフォームブランチを廃止し、これらのプラットフォームに関連付けられた AMI を 2024 年 4 月 10 日に非公開にしました。このアクションにより、Windows Server 2012 環境でデフォルトの Beanstalk AMI を使用するインスタンスが起動されなくなります。

廃止された Windows プラットフォームブランチで実行されている環境がある場合は、最新で完全にサポートされている次の Windows Server プラットフォームのいずれかに移行することをお勧めします。

- IIS 10.0 バージョン 2.x を使用する Windows Server 2022

- IIS 10.0 バージョン 2.x を使用する Windows Server 2019

移行に関するすべての考慮事項については、「Windows サーバープラットフォームの以前のメジャーバージョンからの移行」を参照してください。

プラットフォームの廃止の詳細については、「Elastic Beanstalk プラットフォームのサポートポリシー」を参照してください。

> **Note**
>
> これらの完全にサポートされているプラットフォームに移行できない場合は、Windows Server 2012 R2 または Windows Server 2012 R2 Core の AMI で作成されたカスタム AMI をベースイメージとして使用することを、既に実施していない場合はお勧めします。詳細な手順については、「廃止されたプラットフォームの Amazon マシンイメージ (AMI) へのアクセスを維持する」を参照してください。これらの移行ステップのいずれかを実行中に AMI への一時的なアクセスが必要な場合は、AWS サポートセンターにお問い合わせください。

TLS 1.2 との互換性

2023 年 12 月 31 日をもって、 はすべての AWS API エンドポイントで TLS 1.2 の完全適用 AWS を開始しました。このアクションにより、すべての AWS APIs で TLS バージョン 1.0 および 1.1 を使用する機能が削除されました。この情報はもともと 2022 年 6 月 28 日にお知らせしていました。可用性に影響を与えるリスクを回避するには、ここで特定されているプラットフォームバージョンを実行している環境を、まだ新しいバージョンにアップグレードしていない場合は、できるだけ早くアップグレードしてください。

潜在的な影響

TLS v1.1 以前を実行している Elastic Beanstalk プラットフォームバージョンが影響を受けます。この変更は、設定のデプロイ、アプリケーションのデプロイ、自動スケーリング、新しい環境の起動、ログローテーション、強化されたヘルスレポート、アプリケーションに関連付けられた Amazon S3 バケットへのアプリケーションログの公開などの環境アクションに影響します。

影響を受ける Windows プラットフォームバージョン

以下のプラットフォームバージョンの Elastic Beanstalk 環境をご利用のお客様は、対応する各環境を 2022 年 2 月 18 日にリリースされた Windows プラットフォームバージョン 2.8.3 以降にアップグレードすることをお勧めします。

- Windows Server 2019 — プラットフォームバージョン 2.8.2 またはそれ以前のバージョン

以下のプラットフォームバージョンの Elastic Beanstalk 環境をご利用のお客様は、対応する各環境を 2022 年 12 月 28 日にリリースされた Windows プラットフォームバージョン 2.10.7 以降にアップグレードすることをお勧めします。

- Windows Server 2016 — プラットフォームバージョン 2.10.6 またはそれ以前のバージョン
- Windows Server 2012 – すべてのプラットフォームバージョン。このプラットフォームは 2023 年 12 月 4 日に廃止されました
- Windows Server 2008 — すべてのプラットフォームバージョン。このプラットフォームは 2019 年 10 月 28 日 に廃止されました

最新のサポートされている Windows Server プラットフォームバージョンのリストについては、AWS Elastic Beanstalk プラットフォームガイドの「サポートされているプラットフォーム」を参照してください。

環境の更新に関する詳細とベストプラクティスについては、「Elastic Beanstalk 環境のプラットフォームバージョンの更新」を参照してください。

Elastic Beanstalk を使用した .NET Core (Linux) アプリケーションのデプロイ

> **デ AWS ベロッパーセンターの .NET を確認する**
>
> .Net デベロッパーセンターにアクセスしたことがありますか? これは、.NET on のすべてのモノのワンストップショップです AWS。
> 詳細については、 AWS デベロッパーセンターの .NET を参照してください。

この章では、.NET Core (Linux) ウェブアプリケーションを に設定してデプロイする手順について説明します AWS Elastic Beanstalk。Elastic Beanstalk を使用すると、Amazon Web Services を使用して .NET Core (Linux) ウェブアプリケーションのデプロイ、管理、スケーリングを簡単に行うことができます。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) または Elastic Beanstalk コンソールを使用すると、わずか数分でアプリケーションをデプロイできます。Elastic Beanstalk アプリケーションをデプロイした後も、EB CLI を使用してアプリケーションと環境を管理することも、Elastic Beanstalk コンソール AWS CLIまたは APIs を使用することもできます。

EB CLI を使用して ASP.NET Core の Hello World ウェブアプリケーションを作成およびデプロイするには、「.NET Core on Linux の QuickStart」のステップバイステップの手順に従います。

トピック

- QuickStart: Elastic Beanstalk に .NET Core on Linux アプリケーションをデプロイする
- Elastic Beanstalk 用の .NET Core on Linux 開発環境の設定
- Elastic Beanstalk .NET Core on Linux プラットフォームの使用
- AWS Toolkit for Visual Studio - Elastic Beanstalk での .Net Core の使用
- .NET on Windows Server プラットフォームから Elastic Beanstalk の .NET Core on Linux プラットフォームへの移行

QuickStart: Elastic Beanstalk に .NET Core on Linux アプリケーションをデプロイする

この QuickStart チュートリアルでは、.NET Core on Linux アプリケーションを作成し、 AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

> **Note**
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにはアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: .NET Core on Linux アプリケーションを作成する
- ステップ 2: アプリケーションをローカルに実行する
- ステップ 3: EB CLI を使用して .NET Core on Linux アプリケーションをデプロイする
- ステップ 4: Elastic Beanstalk でアプリケーションを実行する
- ステップ 5：クリーンアップ
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

AWS アカウントをすでにお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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、通話呼び出し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するように求められます。

   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 Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

EB CLI

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

.NET Core on Linux

ローカルマシンに .NET SDK がインストールされていない場合は、.NET ドキュメントウェブサイトの .NET のダウンロードリンクに従ってインストールできます。

次のコマンドを実行して、.NET SDK のインストールを確認します。

```
~$ dotnet --info
```

ステップ 1: .NET Core on Linux アプリケーションを作成する

プロジェクトディレクトリを作成します。

```
~$ mkdir eb-dotnetcore
~$ cd eb-dotnetcore
```

次に、次のコマンドを実行して、サンプルの Hello World アプリケーションを作成します。

```
~/eb-dotnetcore$ dotnet new web --name HelloElasticBeanstalk
~/eb-dotnetcore$ cd HelloElasticBeanstalk
```

ステップ 2: アプリケーションをローカルに実行する

アプリケーションをローカルで実行するには、次のコマンドを実行します。

```
~/eb-dotnetcore/HelloElasticBeasntalk$ dotnet run
```

出力は次のテキストのようになります。

```
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7294
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5052
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
```

> **Note**
>
> `dotnet` コマンドは、アプリケーションをローカルで実行するときに、ポートをランダムに選択します。この例では、ポートは 5052 です。アプリケーションを Elastic Beanstalk 環境にデプロイすると、アプリケーションはポート 5000 で実行されます。

ウェブブラウザに URL アドレス `http://localhost:port` を入力します。この特定の例では、コマンドは `http://localhost:5052` です。ウェブブラウザに「Hello World!」と表示されます。

ステップ 3: EB CLI を使用して .NET Core on Linux アプリケーションをデプロイする

次のコマンドを実行して、このアプリケーションの Elastic Beanstalk 環境を作成します。

環境を作成し、.NET Core on Linux アプリケーションをデプロイするには

1. アプリケーションをコンパイルしてフォルダに公開し、作成しようとしている Elastic Beanstalk 環境にデプロイします。

   ```
   ~$ cd eb-dotnetcore/HelloElasticBeanstalk
   ~/eb-dotnetcore/HelloElasticBeanstalk$ dotnet publish -o site
   ```

2. アプリケーションを公開したばかりのサイトディレクトリに移動します。

   ```
   ~/eb-dotnetcore/HelloElasticBeanstalk$ cd site
   ```

3. eb init コマンドを使用して EB CLI リポジトリを初期化します。

   コマンドで指定するプラットフォームブランチバージョンについては、次の詳細に注意してください。

   - 次のコマンドの *x.y.z* を、プラットフォームブランチ .NET 6 on AL2023 の最新バージョンに置き換えます。
   - 最新のプラットフォームブランチバージョンを特定するには、「AWS Elastic Beanstalk プラットフォーム」ガイドの「.NET Core on Linux でサポートされているプラットフォーム」を参照してください。
   - バージョン番号を含むソリューションスタック名の例は `64bit-amazon-linux-2023-v**3.1.1**-running-.net-6` です。この例では、ブランチバージョンは 3.1.1 です。

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb init -p 64bit-amazon-linux-2023-
vx.y.z-running-.net-6 dotnetcore-tutorial --region us-east-2
Application dotnetcore-tutorial has been created.
```

   このコマンドは、`dotnetcore-tutorial` という名前のアプリケーションを作成し、ローカルリポジトリを設定して、コマンドで指定された .NET Core on Linux プラットフォームバージョンで環境を作成します。

4. (オプション) eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH を使用して connect できるようにします。

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb init
Do you want to set up SSH for your instances?
(y/n): y
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

   1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従ってキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。

5. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。Elastic Beanstalk は、アプリケーションの zip ファイルを自動的にビルドし、ポート 5000 で起動します。

次のように変更します。

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb create dotnet-tutorial
```

Elastic Beanstalk が環境を作成するのに約 5 分かかります。

ステップ 4: Elastic Beanstalk でアプリケーションを実行する

環境を作成するプロセスが完了したら、eb open でウェブサイトを開きます。

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb open
```

お疲れ様でした。Elastic Beanstalk を使用して .NET Core on Linux アプリケーションをデプロイしました。これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ Window が開きます。

ステップ 5：クリーンアップ

アプリケーションでの作業が終了したら、環境を終了できます。Elastic Beanstalk は、環境に関連付けられているすべての AWS リソースを終了します。

EB CLI を使用して Elastic Beanstalk 環境を終了するには、次のコマンドを実行します。

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb terminate
```

AWS アプリケーションの リソース

1 つのインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありません。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリ

プトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。

- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

1 つか 2 つのサンプルアプリケーションをデプロイし、ローカルで .NET Core on Linux アプリケーションを開発して実行する準備が整ったら、「Elastic Beanstalk 用の .NET Core on Linux 開発環境の設定」を参照します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。

Elastic Beanstalk 用の .NET Core on Linux 開発環境の設定

このトピックでは、.NET Core on Linux 開発環境を設定し、アプリケーションを AWS Elastic Beanstalkにデプロイする前にローカルでテストする手順について説明します。また、便利なツールのインストール手順を提供するウェブサイトも参照します。

セクション

- .NET Core SDK のインストール
- IDE のインストール
- のインストール AWS Toolkit for Visual Studio

.NET Core SDK のインストール

.NET Core SDK を使用して、Linux 上で動作するアプリケーションを開発できます。

.NET Core SDK をダウンロードしてインストールするには、.NET ダウンロードページを参照してください。

IDE のインストール

統合された開発環境 (IDE) は、アプリケーション開発を容易にする機能を提供します。.NET 開発に IDE を使用していない場合は、最初に Visual Studio Community をお試しください。

Visual Studio コミュニティページを参照して、Visual Studio Community をダウンロードし、インストールします。

のインストール AWS Toolkit for Visual Studio

AWS Toolkit for Visual Studio は、開発者が を使用して .NET アプリケーションを簡単に開発、デバッグ、デプロイできるようにする Visual Studio IDE 用のオープンソースプラグインです AWS。インストール手順については、「Toolkit for Visual Studio ホームページ」を参照してください。

Elastic Beanstalk .NET Core on Linux プラットフォームの使用

このトピックでは、Elastic Beanstalk で .NET Core on Linux アプリケーションを設定、ビルド、実行する方法について説明します。

AWS Elastic Beanstalk は、Linux オペレーティングシステムで実行されるさまざまな .NET Core フレームワークバージョンの多数のプラットフォームブランチをサポートしています。完全なリストについては、「AWS Elastic Beanstalk プラットフォーム」の「.NET core on Linux」を参照してください。

Elastic Beanstalk Linux ベースのプラットフォームを拡張するさまざまな方法の詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

.NET Core on Linux プラットフォームに関する考慮事項

プロキシサーバー

Elastic Beanstalk .NET Core on Linux プラットフォームには、リクエストをアプリケーションに転送するリバースプロキシが含まれています。デフォルトでは、Elastic Beanstalk はプロキシサーバーとして NGINX を使用します。プロキシサーバーを使用せず、Kestrel をウェブサーバーとして設定できます。Kestrel は、デフォルトで ASP.NET Core プロジェクトテンプレートに含まれています。

アプリケーション構造

Elastic Beanstalk で提供される .NET Core ランタイムを使用するランタイム依存アプリケーションを発行できます。また、ソースバンドルに .NET Core ランタイムとアプリケーションの依存関係を含む自己完結型アプリケーションを公開することもできます。詳細については the section called “アプリケーションのバンドル”を参照してください。

プラットフォーム設定

環境でサーバーインスタンスを実行するプロセスを設定するには、オプションの Procfile をソースバンドルに含めます。`Procfile` は、ソースバンドルにアプリケーションが 1 つ以上ある場合に必要です。

ソースバンドルには、アプリケーションとともに常に `Procfile` を指定することをお勧めします。これにより、アプリケーションに対して Elastic Beanstalk が実行するプロセスを正確に制御できます。

設定オプションは実行中の環境の設定を変更するために Elastic Beanstalk コンソールで利用できます。環境を終了したときにその設定が失われないようにするため、保存済み設定を使用して設定を保存し、それを後で他の環境に適用することができます。

ソースコードの設定を保存する場合、設定ファイルを含めることができます。設定ファイルの設定は、環境を作成するたびに、またはアプリケーションをデプロイするたびに適用されます。設定ファイルを使用して、デプロイの間にパッケージをインストールしたり、スクリプトを実行したり、他のインスタンスのカスタマイズオペレーションを実行することもできます。

Elastic Beanstalk コンソールで適用される設定は、設定ファイルに同じ設定があれば、それらの設定を上書きします。これにより、設定ファイルでデフォルト設定を定義し、コンソールでそのデフォルト設定を環境固有の設定で上書きできます。設定の優先順位の詳細と設定の他の変更方法については、「設定オプション」を参照してください。

.NET Core on Linux 環境の設定

.NET Core on Linux プラットフォームの設定により、Amazon EC2 インスタンスの動作を微調整できます。Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を編集できます。

Elastic Beanstalk コンソールを使用して、Amazon S3 へのログローテーションを有効にでき、アプリケーションが環境から読むことができる変数を設定します。

Elastic Beanstalk コンソールを使用して .NET Core on Linux 環境を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

ログオプション

[Log Options] セクションには、2 つの設定があります。

- インスタンスプロファイル – アプリケーションに関連付けられた Amazon S3 バケットへのアクセス許可が付与されているインスタンスプロファイルを指定します。
- [Enable log file rotation to Amazon S3] (Amazon S3 へのログファイルのローテーションの有効化) - アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーするかどうかを指定します。

環境プロパティ

[Environment Properties (環境プロパティ)] セクションでは、アプリケーションを実行している Amazon EC2 インスタンスの環境設定を指定できます。環境プロパティは、キーバリューのペアでアプリケーションに渡されます。

Elastic Beanstalk 環境変数を実行する .NET Core on Linux 環境内部には、`Environment.GetEnvironmentVariable("variable-name")` を使用してアクセスできます。たとえば、次のコードを使用して変数に `API_ENDPOINT` という名前のプロパティを読み取ることができます。

```
string endpoint = Environment.GetEnvironmentVariable("API_ENDPOINT");
```

詳細については、「環境変数およびその他のソフトウェア設定」を参照してください。

.NET Core on Linux 設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクをパフォーマンスできます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

.NET Core on Linux プラットフォームは、すべての Elastic Beanstalk 環境でサポートされているオプションに加えて、以下の名前空間のオプションをサポートしています。

- `aws:elasticbeanstalk:environment:proxy` - NGINX を使用するか、プロキシサーバーを使用しないかを選択します。有効な値はnginxまたはnoneです。

以下の例の設定ファイルは、.NET Core on Linux 固有の設定オプションの使用を示しています。

Example .ebextensions/proxy-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: none
```

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

.NET Core on Linux Elastic Beanstalk プラットフォーム用のアプリケーションのバンドル

ランタイム依存の .NET Core アプリケーションと、自己完結型の .NET Core アプリケーションの両方を AWS Elastic Beanstalkで実行できます。

ランタイム依存のアプリケーションは、Elastic Beanstalk がアプリケーションを実行するために提供する .NET Core ランタイムを使用します。Elastic Beanstalk は、ソースバンドル内の `runtimeconfig.json` ファイルに基づいて、アプリケーションに使用するランタイムを決定します。Elastic Beanstalk は、アプリケーションに使用できる最新の互換性のあるランタイムを選択します。

自己完結型アプリケーションには、.NET Core ランタイム、アプリケーション、およびその依存関係が含まれます。Elastic Beanstalk でプラットフォームに含まれない .NET Core ランタイムのバージョンを使用するには、自己完結型アプリケーションを提供します。

例

`dotnet publish` コマンドを使用すると、自己完結型アプリケーションとランタイム依存アプリケーショcompileンの両方をコンパイルできます。.NET Core アプリの発行の詳細については、.NET Core ドキュメントの「.NET Core アプリケーションの発行の概要」を参照してください。

以下のファイル構造例では、Elastic Beanstalk が提供する .NET Core ランタイムを使用する単一のアプリケーションを定義します。

```
### appsettings.Development.json
### appsettings.json
### dotnetcoreapp.deps.json
### dotnetcoreapp.dll
### dotnetcoreapp.pdb
### dotnetcoreapp.runtimeconfig.json
### web.config
### Procfile
### .ebextensions
### .platform
```

ソースバンドルには複数のアプリケーションを含めることができます。次の例では、同じウェブサーバー上で実行する 2 つのアプリケーションを定義します。複数のアプリケーションを実行するには、ソースバンドルに Procfile を含める必要があります。完全なサンプルアプリケーションについては、dotnet-core-linux-multiple-apps.zip を参照してください。

```
### DotnetMultipleApp1
#   ### Amazon.Extensions.Configuration.SystemsManager.dll
#   ### appsettings.Development.json
#   ### appsettings.json
#   ### AWSSDK.Core.dll
#   ### AWSSDK.Extensions.NETCore.Setup.dll
#   ### AWSSDK.SimpleSystemsManagement.dll
#   ### DotnetMultipleApp1.deps.json
#   ### DotnetMultipleApp1.dll
#   ### DotnetMultipleApp1.pdb
#   ### DotnetMultipleApp1.runtimeconfig.json
#   ### Microsoft.Extensions.PlatformAbstractions.dll
#   ### Newtonsoft.Json.dll
#   ### web.config
### DotnetMultipleApp2
#   ### Amazon.Extensions.Configuration.SystemsManager.dll
#   ### appsettings.Development.json
#   ### appsettings.json
#   ### AWSSDK.Core.dll
#   ### AWSSDK.Extensions.NETCore.Setup.dll
#   ### AWSSDK.SimpleSystemsManagement.dll
#   ### DotnetMultipleApp2.deps.json
#   ### DotnetMultipleApp2.dll
#   ### DotnetMultipleApp2.pdb
#   ### DotnetMultipleApp2.runtimeconfig.json
#   ### Microsoft.Extensions.PlatformAbstractions.dll
#   ### Newtonsoft.Json.dll
#   ### web.config
### Procfile
### .ebextensions
### .platform
```

Procfile を使用した .NET Core on Linux Elastic Beanstalk 環境の設定

同じウェブサーバーで複数のアプリケーションを実行するには、実行するアプリケーションを Elastic Beanstalk に指示する `Procfile` をソースバンドルに含める必要があります。

ソースバンドルには、アプリケーションとともに常に `Procfile` を指定することをお勧めします。このようにして、アプリケーションに対して Elastic Beanstalk が実行するプロセスと、これらのプロセスが受け取る引数を正確に制御できます。

以下の例では、`Procfile` を使用して、Elastic Beanstalk が同じウェブサーバーで実行する 2 つのアプリケーションを指定します。

Example [Procfile]

```
web: dotnet ./dotnet-core-app1/dotnetcoreapp1.dll
web2: dotnet ./dotnet-core-app2/dotnetcoreapp2.dll
```

`Procfile` 書き込みと使用の詳細については、「ビルドファイルと Procfile」を参照してください。

プロキシサーバーを設定します

AWS Elastic Beanstalk はリバースプロキシとして NGINX を使用して、アプリケーションにリクエストを中継します。Elastic Beanstalk では、デフォルトの NGINX 設定が用意されています。これは、独自の設定で拡張することも完全に上書きすることもできます。

デフォルトでは、Elastic Beanstalk はポート 5000 でアプリケーションにリクエストを転送するように NGINX プロキシを設定します。デフォルトのポートを上書きするには、`PORT` 環境プロパティを、主要なアプリケーションがリッスンするポートに設定します。

> **Note**
>
> アプリケーションがリッスンしているポートは、ロードバランサーからリクエストを受信するために NGINX サーバーがリッスンするポートに影響を与えません。

ご使用のプラットフォームバージョンでプロキシサーバーを設定する

すべての AL2023/AL2 プラットフォームでは、統一されたプロキシ設定機能がサポートされています。AL2023/AL2 を実行中のプラットフォームバージョンでプロキシサーバーを設定する方法の詳細については、「リバースプロキシの設定」を参照してください。

次の設定ファイルの例では、環境の NGINX 設定を拡張します。この設定は、ウェブサーバーのポート 5200 でリッスンする 2 番目のウェブアプリケーションに、`/api` へのリクエストを転送します。デフォルトでは、Elastic Beanstalk はポート 5000 でリッスンする 1 つのアプリケーションにリクエストを転送します。

Example **`01_custom.conf`**

```
location /api {
```

```
    proxy_pass          http://127.0.0.1:5200;
    proxy_http_version  1.1;

    proxy_set_header   Upgrade $http_upgrade;
    proxy_set_header   Connection $http_connection;
    proxy_set_header   Host $host;
    proxy_cache_bypass $http_upgrade;
    proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header   X-Forwarded-Proto $scheme;
}
```

AWS Toolkit for Visual Studio - Elastic Beanstalk での .Net Core の使用

このトピックでは、 AWS Toolkit for Visual Studioを使用して以下のタスクを実行する方法を示します

- Visual Studio テンプレートを使用して ASP .NET Core ウェブアプリケーションを作成します。
- Elastic Beanstalk Amazon Linux 環境を作成する。
- ASP.NET Core ウェブアプリケーションを新しい Amazon Linux 環境にデプロイします。

このトピックでは、 を使用して Elastic Beanstalk アプリケーション環境 AWS Toolkit for Visual Studio を管理し、アプリケーションの状態をモニタリングする方法についても説明します。

AWS Toolkit for Visual Studio は Visual Studio IDE へのプラグインです。このツールキットを使用すると、Visual Studio 環境での作業中に、Elastic Beanstalk でアプリケーションをデプロイおよび管理できます。

セクション

- 前提条件
- 新しいアプリケーションプロジェクトの作成
- Elastic Beanstalk 環境を作成してアプリケーションをデプロイする
- 環境を終了する
- Elastic Beanstalk アプリケーション環境の管理
- アプリケーションの状態をモニタリングする

前提条件

このチュートリアルを開始する前に、 AWS Toolkit for Visual Studioをインストールする必要があります 手順については、「AWS Toolkit for Visual Studioのセットアップ」を参照してください。

ツールキットを使用したことがない場合は、ツールキットのインストール後に最初に行う必要があるのは、ツールキットに AWS 認証情報を登録することです。詳細については、「AWS 認証情報の提供」を参照してください。

新しいアプリケーションプロジェクトの作成

Visual Studio に .NET Core アプリケーションプロジェクトがない場合は、いずれかの Visual Studio プロジェクトテンプレートを使用して簡単に作成できます。

新しい ASP.NET Core ウェブアプリケーションプロジェクトを作成するには

1. Visual Studio の [File (ファイル)] メニューで [New (新規)] を選択し、[Project (プロジェクト)] を選択します。
2. [新規プロジェクトの作成] ダイアログボックスで、[C#]、[Linux]、[クラウド] の順に選択します。
3. 表示されるプロジェクトテンプレートのリストから [ASP.NET Core ウェブアプリケーション] を選択し、[次へ] を選択します。

> **Note**
>
> プロジェクトテンプレートに [ASP.NET Core Web Application (ASP.NET Core ウェブアプリケーション)] が表示されない場合は、Visual Studio でインストールできます。
>
> 1. テンプレートリストの 1 番下までスクロールし、テンプレートリストの下にある [Install more tools and features (その他のツールと機能のインストール)] リンクを選択します。
> 2. Visual Studio アプリケーションにデバイスに対する変更を許可するように求められた場合は、[Yes (はい)] を選択します。
> 3. [ワークロード] タブを選択し、[ASP.NET and web development (ASP.NET とウェブ開発)] を選択します。
> 4. [Modify (変更)] ボタンを選択します。Visual Studio インストーラによってプロジェクトテンプレートがインストールされます。

5. インストーラの完了後、パネルを終了して、Visual Studio で終了した場所に戻ります。

4. [Configure your new project (新しいプロジェクトの設定)] ダイアログボックスで、[Project name (プロジェクト名)] に入力します。[Solution name (ソリューション名)] はデフォルトでプロジェクト名になります。次に、[作成] を選択します。
5. [Create a new ASP.NET Core web application (新しい ASP.NET Core ウェブアプリケーションの作成)] ダイアログボックスで、[.NET Core]、[ASP.NET Core 3.1] の順に選択します。表示されたアプリケーションタイプのリストから [Web Application (ウェブアプリケーション)] を選択してから、[Create (作成)] ボタンを選択します。

Visual Studio では、アプリケーションの作成時に [Creating Project (プロジェクトの作成中)] ダイアログボックスが表示されます。Visual Studio でアプリケーションの生成が完了すると、パネルにアプリケーション名が表示されます。

Elastic Beanstalk 環境を作成してアプリケーションをデプロイする

このセクションでは、アプリケーションの Elastic Beanstalk 環境を作成し、その環境にアプリケーションをデプロイする方法について説明します。

新しい環境を作成してアプリケーションをデプロイするには

1. Visual Studio で、[View (表示)]、[Solution Explorer (ソリューションエクスプローラー)] の順に選択します。
2. [ソリューションエクスプローラー] で、アプリケーションのコンテキストメニュー (右クリック) を開き、[AWS Elastic Beanstalkへの発行] を選択します。
3. [に発行] AWS Elastic Beanstalkウィザードで、アカウント情報を入力します。
   a. [Account profile to use (使用するアカウントのプロファイル)] で、[既定] のアカウントを選択するか、[Add another account (別のアカウントを追加)] アイコンを選択して新しいアカウント情報を入力します。
   b. [Region (リージョン)] で、アプリケーションをデプロイするリージョンを選択します。利用可能な AWS リージョンの詳細については、AWS Elastic Beanstalk 「」の「エンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。Elastic Beanstalk でサポートされていないリージョンを選択した場合、Elastic Beanstalk にデプロイするオプションは使用できません。

c. [Create a new application environment (新しいアプリケーション環境を作成する)] を選択し、[次へ] を選択します。

4. [アプリケーション環境] ダイアログボックスで、新しいアプリケーション環境の詳細を入力します。

5. 次のAWSオプションダイアログボックスで、デプロイされたアプリケーションの Amazon EC2 オプションおよびその他の AWS 関連オプションを設定します。

   a. [コンテナタイプ] で、[64bit Amazon Linux 2 v*<n.n.n>* running .NET Core] を選択します。

   > **Note**
   >
   > Linux の最新のプラットフォームバージョンを選択することをお勧めします。このバージョンには、最新の Amazon マシンイメージ (AMI) に含まれている最新のセキュリティとバグ修正が含まれています。

   b. [インスタンスタイプ] で、[t2.マイクロ] を選択します (マイクロインスタンスタイプを選択すると、インスタンスの実行に関連するコストが最小限に抑えられます)。

   c. [Key pair] で [Create new key pair (新しいキーペアの作成)] を選択します。新しいキーペアの名前を入力し、[OK] を選択します。(この例では、**myuseastkeypair** を使用します)。キーペアを使用すると、Amazon EC2 インスタンスへのリモートデスクトップアクセスが可能になります。Amazon EC2 キーペアの詳細については、「Amazon Elastic Compute Cloud ユーザーガイド」の「認証情報の使用」を参照してください。

   d. シンプルでトラフィックの少ないアプリケーションの場合は、[Single instance environment (シングルインスタンス環境)] を選択します。詳細については、「環境タイプ」を参照してください。

   e. [次へ] を選択してください。

   この例で使用されていない AWS オプションの詳細については、次のページを参照してください。

   - [Use custom AMI (カスタム AMI の使用)] については、「Elastic Beanstalk 環境でのカスタム Amazon マシンイメージ (AMI) の使用」を参照してください。
   - [Single instance environment (シングルインスタンス環境)] を選択しない場合は、[Load balance type (ロードバランスのタイプ)] を選択する必要があります。詳細については「Elastic Beanstalk 環境のロードバランサー」を参照してください。

- [Use non-default VPC (デフォルト以外の VPC の使用)] を選択しなかった場合、Elastic Beanstalk はデフォルトの Amazon VPC (Amazon Virtual Private Cloud) の設定を使用します。詳細については、「Amazon VPC で Elastic Beanstalk を使用する」を参照してください。
- [Enable Rolling Deployments (ローリングデプロイを有効にする)] オプションを選択すると、デプロイがバッチに分割され、デプロイ中に発生する可能性のあるダウンタイムが回避されます。詳細については、「Elastic Beanstalk 環境へのアプリケーションのデプロイ」を参照してください。
- [Relational Database Access (リレーショナルデータベースアクセス)] オプションを選択すると、Elastic Beanstalk 環境が、Amazon RDS DB セキュリティグループを使用して以前に作成した Amazon RDS データベースに接続されます。詳細については、Amazon RDS ユーザーガイドのセキュリティグループによるアクセスのコントロールを参照してください。

6. [許可] ダイアログボックスで [次へ] を選択します。
7. [アプリケーションオプション] ダイアログボックスで [次へ] を選択します。
8. デプロイのオプションを確認します。設定が正しいことを確認したら、[Deploy (デプロイ)] を選択します。

ASP.NET Core ウェブアプリケーションはウェブデプロイファイルとしてエクスポートされます。このファイルは Amazon S3 にアップロードされ、Elastic Beanstalk により新しいアプリケーションバージョンとして登録されます。Elastic Beanstalk デプロイ機能は、環境を新しくデプロイされたコードで利用できるようになるまでモニタリングします。[Env:<environment name>] タブに、環境の [Status (ステータス)] が表示されます。ステータスが [Environment is healthy (環境は正常です)] に更新されたら、ウェブアプリケーションを起動する URL アドレスを選択できます。

環境を終了する

未使用の AWS リソースに対して料金が発生しないようにするには、 AWS Toolkit for Visual Studio を使用して実行中の環境を終了できます。

> **Note**
> いつでも、また同じバージョンを使用して新しい環境を起動できます。

環境を終了するには

1. Elastic Beanstalk ノードとアプリケーションノードを展開します。AWS Explorer で、アプリケーション環境のコンテキストメニュー (右クリック) を開き、[環境の終了] を選択します。
2. プロンプトが表示されたら、[はい] を選択して、環境を終了することを確認します。Elastic Beanstalk が環境で実行されている AWS リソースを終了するまでに数分かかります。

[Env:<environment name>] タブで、環境の [Status (ステータス)] が [Terminating (終了中)] に変わり、最終的に [Terminated (終了)] に変わります。

> **Note**
>
> 環境を終了すると、終了した環境に関連付けられていた CNAME はすべてのユーザーが使用できるようになります。

Elastic Beanstalk アプリケーション環境の管理

このセクションでは、アプリケーション環境設定の一部として AWS Toolkit for Visual Studio で編集できる特定のサービス設定について説明します。 AWS Toolkit for Visual Studio と AWS マネジメントコンソールを使用すると、アプリケーション環境で使用される AWS リソースのプロビジョニングと設定を変更できます。 AWS マネジメントコンソールを使用してアプリケーション環境を管理する方法については、「」を参照してくださいElastic Beanstalk での環境の作成。

環境設定を変更します

アプリケーションをデプロイすると、Elastic Beanstalk は接続された複数の AWS クラウドコンピューティングサービスを設定します。 AWS Toolkit for Visual Studio を使用して、これらの個々のサービスの設定方法を制御できます。

アプリケーションの環境設定を編集するには

1. Visual Studio の [ファイル] メニューで、[AWS Explorer] を選択します。
2. Elastic Beanstalk ノードとアプリケーションノードを展開します。アプリケーション環境のコンテキスト (右クリック) メニューを開き、View Status (ステータスの表示) を選択します。

   次の設定を編集できます。

   - AWS X-Ray

- [Server] (サーバー)
- ロードバランサー (複数インスタンス環境にのみ適用)
- Auto Scaling (複数インスタンス環境にのみ適用)
- 通知
- コンテナ
- アドバンスト設定オプション

Toolkit AWS for Visual Studio を使用した AWS X-Ray の設定

AWS X-Ray は、リクエストトレース、例外収集、プロファイリング機能を提供します。 AWS X-Ray パネルを使用すると、アプリケーションの X-Ray を有効または無効にできます。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。

AWS Toolkit for Visual Studio を使用した EC2 インスタンスの設定

Amazon Elastic Compute Cloud (Amazon EC2) を使用して、Amazon のデータセンターでサーバーインスタンスを起動および管理できます。必要に応じて、任意のリーガルの用途で Amazon EC2 サーバーインスタンスをいつでも使用できます。インスタンスはさまざまなサイズや設定で使用できます。詳細については、「Amazon EC2」を参照してください。

AWS Toolkit for Visual Studio のアプリケーション環境タブ内の [サーバー] タブで Amazon EC2 インスタンス設定を編集できます。

Amazon EC2 インスタンスタイプ

[インスタンスタイプ] には、Elastic Beanstalk アプリケーションで使用できるインスタンスタイプが表示されます。インスタンスタイプを変更して、アプリケーションに最適な特性 (メモリサイズや CPU 能力など) を持つサーバーを選択します。例えば、高負荷の操作を長時間実行するアプリケーションでは、より大きい CPU やメモリが必要となる場合があります。

Elastic Beanstalk アプリケーションで使用可能な Amazon EC2 インスタンスタイプの詳細については、Amazon Elastic Compute Cloud ユーザーガイドの「インスタンスタイプ」を参照してください。

Amazon EC2 セキュリティグループ

Amazon EC2 セキュリティグループを使用すると、Elastic Beanstalk アプリケーションへのアクセスを制御できます。セキュリティグループとは、インスタンスのファイアウォールのルールを定義するものです。このルールでは、どの着信ネットワークトラフィックをインスタンスに配信するかを指定します。他のすべての受信トラフィックは破棄されます。グループのルールはいつでも変更することができます。実行中のすべてのインスタンスと、今後起動されるインスタンスについて、新しいルールが自動的に実施されます。

Elastic Beanstalk アプリケーションへのアクセスをコントロールする Amazon EC2 セキュリティグループを指定できます。そのためには、特定の Amazon EC2 セキュリティグループの名前 (複数のセキュリティグループの名前はカンマで区切る) を [EC2 Security Groups (EC2 セキュリティグループ)] テキストボックスに入力します。これを行うには、 AWS マネジメントコンソールまたは AWS Toolkit for Visual Studio を使用します。

AWS Toolkit for Visual Studio を使用してセキュリティグループを作成するには

1. Visual Studio の [AWS Explorer] で [Amazon EC2] ノードを展開し、[セキュリティグループ] を選択します。
2. [Create Security Group (セキュリティグループの作成)] を選択し、セキュリティグループの名前と説明を入力します。
3. [OK] を選択します。

Amazon EC2 セキュリティグループの詳細については、Amazon Elastic Compute Cloud ユーザーガイドの「セキュリティグループの使用」を参照してください。

Amazon EC2 のキーペア

Elastic Beanstalk アプリケーション用にプロビジョニングされた Amazon EC2 インスタンスには、Amazon EC2 のキーペアを使用して安全にログインできます。

> **Important**
>
> Amazon EC2 のキーペアを作成した後、これらのインスタンスにアクセスできるように、Elastic Beanstalk によってプロビジョニングされた Amazon EC2 インスタンスを設定する必要があります。アプリケーションを Elastic Beanstalk にデプロイするときに、 AWS Toolkit for Visual Studio 内の Publish to AWS ウィザードを使用してキーペアを作成できます。ツールキット を使用してさらにキーペアを作成する場合は、ここで説明している手順

に従ってください。または、AWS マネジメントコンソールを使用して、Amazon EC2 のキーペアを設定することもできます。Amazon EC2 のキーペアを作成する手順については、Amazon Elastic Compute Cloud 入門ガイドを参照してください。

[Existing Key Pair (既存のキーペア)] テキストボックスを使用すると、Elastic Beanstalk アプリケーションを実行中の Amazon EC2 インスタンスに安全にログインするために使用できる Amazon EC2 のキーペアの名前を指定できます。

Amazon EC2 のキーペアの名前を指定するには

1. [Amazon EC2] ノードを展開し、[キーペア] を選択します。
2. [Create Key Pair (キーペアの作成)] を選択し、キーペア名を入力します。
3. [OK] を選択します。

Amazon EC2 のキーペアの詳細については、Amazon Elastic Compute Cloud ユーザーガイドの「Amazon EC2 認証情報の使用」のページを参照してください。Amazon EC2 インスタンスへの接続の詳細については、以下を参照してください。

間隔のモニタリング

デフォルトでは、基本的な Amazon CloudWatch メトリクスだけが有効化されています。5 分周期でデータを返します。 AWS Toolkit for Eclipseの環境の [Configuration] (設定) タブの [Server] (サーバー) セクションで、[Monitoring Interval] (モニタリング間隔) で [1 minute] (1 分)を選択すると、より詳細な 1 分間隔の CloudWatch メトリクスを有効化することもできます。

> **Note**
>
> Amazon CloudWatch の利用料金で 1 分間隔のメトリクスに申し込むことができます。詳細については、Amazon CloudWatch を参照してください。

カスタム AMI ID

AWS Toolkit for Eclipseの環境で、[Configuration] (設定) タブの [Server] (サーバー) セクションの [Custom AMI ID] (カスタム AMI ID) ボックスにカスタム AMI の ID を入力して、Amazon EC2 インスタンスで使用するデフォルトの AMI を独自のカスタム AMI に置き換えることができます。

> **Important**
>
> 独自の AMI の使用は高度な作業であるため、注意が必要です。カスタム AMI が必要な場合は、デフォルトの Elastic Beanstalk AMI を変更して使用することをお勧めします。正常と見なされるには、Amazon EC2 インスタンスが、ホストマネージャの実行を含む Elastic Beanstalk の一連の要件を満たす必要があります。これらの要件を満たさないと、環境が正常に動作しない可能性があります。

AWS ツールキット for Visual Studio を使用した Elastic Load Balancing の設定

Elastic Load Balancing は、アプリケーションの可用性とスケーラビリティの向上に役立つ Amazon のウェブサービスです。このサービスによって、アプリケーションの負荷を簡単に複数の Amazon EC2 インスタンスに配信できます。Elastic Load Balancing による冗長化で可用性が向上し、アプリケーションのトラフィック増加を suppot できます。

Elastic Load Balancing を使用すると、すべての実行中のインスタンス間で受信アプリケーショントラフィックを自動的に配信してバランスをとることができます。アプリケーションの容量を増やす必要がある場合は、新しいインスタンスを簡単に追加することもできます。

アプリケーションをデプロイすると、Elastic Beanstalk によって自動的に Elastic Load Balancing がプロビジョニングされます。 AWS Toolkit for Visual Studio のアプリケーション環境タブ内の Load Balancer タブを使用して、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を編集できます。

ここでは、アプリケーションで設定できる Elastic Load Balancing パラメータについて説明します。

ポート

Elastic Beanstalk アプリケーションへのリクエストを処理するためにプロビジョニングされたロードバランサーは、アプリケーションを実行している Amazon EC2 インスタンスにリクエストを送信します。プロビジョニングされたロードバランサーは、HTTP ポートと HTTPS ポートでリクエストをリッスンし、 AWS Elastic Beanstalk アプリケーション内の Amazon EC2 インスタンスにリクエストをルーティングできます。デフォルトでは、ロードバランサーは HTTP ポートのリクエストを処理します。そのためには、1 つ以上のポート (HTTP または HTTPS) を有効にする必要があります。

> **Important**
> 指定したポートがロックされていないことを確認してください。ロックされている場合は、Elastic Beanstalk アプリケーションに接続できません。

HTTP ポートをコントロールします

HTTP ポートをオフにするには、[HTTP Listener Port] で [OFF] を選択します。HTTP ポートを有効にするには、リストから HTTP ポート ([80] など) を選択します。

> **Note**
> デフォルトポート 80 以外のポート (例: ポート 8080) を使用して環境にアクセスする場合は、既存のロードバランサーにリスナーを追加し、そのポートでリッスンするようにリスナーを設定します。
> 例えば、Classic Load Balancer 用のAWS CLI を使用して、次のコマンドを入力します。*LOAD_BALANCER_NAME* は Elastic Beanstalk のロードバランサーの名前に置き換えてください。
>
> ```
> aws elb create-load-balancer-listeners --load-balancer-name LOAD_BALANCER_NAME
> --listeners "Protocol=HTTP, LoadBalancerPort=8080, InstanceProtocol=HTTP,
> InstancePort=80"
> ```
>
> 例えば、Application Load Balancer 用のAWS CLI を使用して、次のコマンドを入力します。*LOAD_BALANCER_ARN* は Elastic Beanstalk のロードバランサーの ARN に置き換えてください。
>
> ```
> aws elbv2 create-listener --load-balancer-arn LOAD_BALANCER_ARN --protocol HTTP
> --port 8080
> ```
>
> Elastic Beanstalk を使用して環境を監視する場合は、ポート 80 のリスナーを削除しないでください。

HTTPS ポートを制御する

Elastic Load Balancing は、ロードバランサーへのクライアント接続のトラフィックを暗号化するために、HTTPS/TLS プロトコルをサポートしています。ロードバランサーから EC2 インスタンスへ

の接続では、プレーンテキストの暗号化が使用されます。デフォルトで、HTTPS ポートは無効です。

HTTPS ポートを有効にするには

1. AWS Certificate Manager (ACM) を使用して新しい証明書を作成するか、証明書とキーを AWS Identity and Access Management (IAM) にアップロードします。ACM 証明書のリクエストの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のリクエスト」を参照してください。ACM へのサードパーティー証明書のインポートの詳細については、AWS Certificate Manager ユーザーガイドの「証明書のインポート」を参照してください。ACM がお客様のリージョンで利用できない場合は、(IAM) を使用してサードパーティー証明書をアップロードします。 AWS Identity and Access Management ACM および IAM サービスは証明書を保存し、SSL 証明書の Amazon リソースネーム (ARN) を提供します。証明書の作成と IAM へのアップロードに関する詳細については、IAM ユーザーガイドの「サーバー証明書の使用」を参照してください。
2. [HTTPS Listener Port (HTTPS リスナーポート)] のポートを選択して、HTTPS ポートを指定します。
3. [SSL 証明書 ID] に、SSL 証明書の Amazon リソースネーム (ARN) を入力します。例えば、**`arn:aws:iam::123456789012:server-certificate/abc/certs/build`**、**`arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`** などです。ステップ 1 で作成またはアップロードした SSL 証明書を使用します。

HTTPS ポートをオフにするには、[HTTPS Listener Port] で [OFF] を選択します。

ヘルスチェック

ヘルスチェックの定義には、インスタンスのヘルスを照会する URL が含まれます。デフォルトでは、Elastic Beanstalk はレガシーではないコンテナの場合は TCP:80 を使用し、レガシーコンテナの場合は HTTP:80 を使用します。デフォルト URL を上書きしてアプリケーションの既存のリソースに一致させるには (`/myapp/default.aspx` などにするには)、その URL を [Application Health Check URL (アプリケーションヘルスチェックの URL)] ボックスに入力します。デフォルトの URL をオーバーライドすると、Elastic Beanstalk は HTTP を使用してリソースを照会します。レガシーコンテナタイプを使用しているかどうかを確認するには、「the section called “一部のプラットフォームバージョンがレガシーとマークされているのはなぜですか?”」を参照してください。

[Load Balancing] パネルの [EC2 Instance Health Check] を使用して、ヘルスチェックの設定を制御できます。

ヘルスチェックの定義には、インスタンスのヘルスをクエリする URL が含まれます。デフォルト URL を上書きしてアプリケーションの既存のリソースに一致させるには (`/myapp/index.jsp` などにするには)、その URL を [Application Health Check URL (アプリケーションヘルスチェックの URL)] ボックスに入力します。

次の一覧では、アプリケーションに設定できるヘルスチェックパラメータについて説明します。

- [Health Check Interval (seconds)] には、Elastic Load Balancing がアプリケーションの Amazon EC2 インスタンスの各ヘルスチェックを待機する秒数を入力します。
- [Health Check Timeout (seconds)] には、Elastic Load Balancing がインスタンスの応答がないとみなす応答待機時間の秒数を入力します。
- [Healthy Check Count Threshold] および [Unhealthy Check Count Threshold] には、Elastic Load Balancing がインスタンスのヘルスステータスを変更するまでの URL 探索の連続成功回数または失敗回数を指定します。たとえば、[Unhealthy Check Count Threshold (ヘルスチェック失敗数のしきい値)] ボックスで **5** を指定した場合、Elastic Load Balancing がヘルスチェックを失敗とみなすには、URL からエラーメッセージまたはタイムアウトが 5 回連続して返される必要があります。

セッション

デフォルトでは、ロードバランサーは負荷が最小になるように、各リクエストを個別にサーバーインスタンスにルーティングします。比較すると、スティッキーセッションの場合、セッション中にユーザーから受信するすべてのリクエストが、同じサーバーインスタンスに送信されるように、ユーザーのセッションを特定のサーバーインスタンスにバインドします。

アプリケーションでスティッキーセッションが有効な場合、Elastic Beanstalk は、ロードバランサーで生成された HTTP Cookie を使用します。ロードバランサーは、ロードバランサーが生成する特別な Cookie を使って、各リクエストのアプリケーションインスタンスを追跡します。ロードバランサーがリクエストを受け取ると、まずこの Cookie がリクエスト内にあるかどうかを調べます。その Cookie がある場合、リクエストは Cookie で指定されたアプリケーションインスタンスに送信されます。Cookie がない場合、ロードバランサーは、既存のロードバランシングアルゴリズムに基づいてアプリケーションインスタンスを選択します。同じユーザーからの以降のリクエストをそのアプリケーションインスタンスにバインドするため、応答に Cookie が挿入されます。ポリシー設定では、各 Cookie の有効期間を設定する Cookie 期限を検証します。

[Load Balancer (ロードバランサー)] タブの [Sessions (セッション)] セクションを使用して、アプリケーションのロードバランサーでスティッキーセッションの使用を許可できるようにするかどうかを指定できます。

Elastic Load Balancing の詳細については、Elastic Load Balancing デベロッパーガイドを参照してください。

Toolkit AWS for Visual Studio を使用した Auto Scaling の設定

Amazon EC2 Auto Scaling は、ユーザーが定義したトリガーに基づいて、Amazon EC2 インスタンスを自動的に起動または終了するように設計された Amazon のウェブサービスです。ユーザーは Auto Scaling グループをセットアップし、そのグループにトリガーを関連付けることで、帯域幅の使用や CPU の使用率などのメトリクスに基づいて、コンピューティングリソースを自動的にスケーリングできます。Amazon EC2 Auto Scaling は Amazon CloudWatch と連携して、アプリケーションを実行するサーバーインスタンスのメトリクスを取得します。

Amazon EC2 Auto Scaling によって、Amazon EC2 インスタンスのグループを利用して、自動的に数を増減できるようにさまざまなパラメータを設定できます。Amazon EC2 Auto Scaling は、アプリケーションのトラフィックの変化をシームレスに処理できるように、Amazon EC2 インスタンスのグループを追加または削除できます。

Amazon EC2 Auto Scaling は、起動した各 Amazon EC2 インスタンスの状態もモニタリングします。インスタンスが予期せずに終了した場合、Amazon EC2 Auto Scaling は終了を検出し、代わりのインスタンスを起動します。この機能を使用すると、任意の固定の Amazon EC2 インスタンス数を自動的に維持できます。

Elastic Beanstalk はアプリケーション用に Amazon EC2 Auto Scaling のプロビジョニングを行います。 AWS Toolkit for Visual Studio のアプリケーション環境タブ内の Auto Scaling タブを使用して、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を編集できます。

ここでは、アプリケーションの Auto Scaling パラメータの設定方法について説明します。

設定の起動

起動設定を編集すると、Elastic Beanstalk アプリケーションによる Amazon EC2 Auto Scaling リソースのプロビジョニング方法を制御できます。

[Minimum Instance Count (最小インスタンス数)] ボックスと [Maximum Instance Count (最大インスタンス数)] ボックスを使用して、Elastic Beanstalk アプリケーションが使用する Auto Scaling グループの最小サイズと最大サイズを指定できます。

> **Note**
>
> 固定の Amazon EC2 インスタンス数を維持するには、[Minimum Instance Count] と [Maximum Instance Count] を同じ値に設定します。

[Availability Zones] ボックスでは、Amazon EC2 インスタンスを維持するアベイラビリティーゾーンの数を指定できます。耐障害性アプリケーションを構築する場合、この数を設定することをお勧めします。1 つのアベイラビリティーゾーンが停止しても、インスタンスは他のアベイラビリティーゾーンで実行されます。

> **Note**
>
> 現在、インスタンスを維持するアベイラビリティーゾーンを指定することはできません。

トリガー

トリガーとは、インスタンス数を増やす (スケールアウト) タイミングや、インスタンス数を減らす (スケールイン) タイミングをシステムに指示するために設定する Amazon EC2 Auto Scaling のメカニズムです。CPU の使用率など、Amazon CloudWatch に発行された任意のメトリクスについて、トリガーが発生するように設定し、指定した条件を満たしているかどうかを判断できます。メトリクスについて指定した条件の上限または下限を、指定した期間超過すると、トリガーによって Scaling Activity という長時間実行されるプロセスが起動されます。

AWS Toolkit for Visual Studio を使用して、Elastic Beanstalk アプリケーションのスケーリングトリガーを定義できます。

Amazon EC2 Auto Scaling トリガーは、特定のインスタンスの特定の Amazon CloudWatch メトリクスをモニタリングすることで機能します。メトリクスには、CPU 使用率、ネットワークトラフィック、ディスクアクティビティなどがあります。Trigger Measurement 設定を使用して、トリガーのメトリクスを選択します。

次のリストでは、 AWS マネジメントコンソールを使用して設定できるトリガーパラメータについて説明します。

- トリガーに使用する統計を指定できます。[Trigger Statistic] に対して選択できるのは、[Minimum]、[Maximum]、[Sum]、または [Average] です。
- [Unit of Measurement] には、トリガー測定の単位を指定します。
- Measurement Period ボックスの値は、Amazon CloudWatch がトリガーのメトリクスを計測する頻度を指定します。Breach Duration (違反期間) は、トリガーが発生するまでに、定義した限度 ([Upper Threshold (上限)] と [Lower Threshold (下限)] に指定した値) をメトリックが超過できる時間を示します。
- 上限超過スケール増分 と下限超過スケール増分 には、規模の拡大や縮小を実行する際に追加または削除する Amazon EC2 インスタンスの数を指定します。

Amazon EC2 Auto Scaling の詳細については、Amazon Elastic Compute Cloud ドキュメントの「Amazon EC2 Auto Scaling」セクションを参照してください。

AWS Toolkit for Visual Studio を使用した通知の設定

Elastic Beanstalk では、Amazon Simple Notification Service (Amazon SNS) を使用して、アプリケーションに影響を与える重要なイベントについて通知します。Amazon SNS 通知を有効化するには、[Email Address (E メールアドレス)] ボックスに E メールアドレスを入力します。Amazon SNS 通知を無効にするには、ボックスから電子メールアドレスを削除します。

AWS Toolkit for Visual Studio を使用した追加の環境オプションの設定

Elastic Beanstalk は、環境の動作、およびその環境に含まれるリソースの設定に使用する多数の設定オプションを定義します。設定オプションは、`aws:autoscaling:asg` のような名前空間に編成されています。名前空間はそれぞれ、環境の Auto Scaling グループのオプションを定義します。Advanced (アドバンスト) パネルには、環境の作成後に更新できる設定オプションの名前空間がアルファベット順に一覧表示されます。

各項目のデフォルト値やサポートされる値を含む名前空間とオプションの完全なリストについては、すべての環境に対する汎用オプションとLinux プラットフォームオプション上の .NET Coreを参照してください。

Toolkit AWS for Visual Studio を使用した .NET Core コンテナの設定

コンテナ パネルでは、アプリケーションコードから読み取ることができる環境変数を指定できます。

アプリケーションの状態をモニタリングする

このトピックでは、アプリケーションのウェブサイトの状態をモニタリングする方法について説明します。本稼働ウェブサイトが利用可能であり、リクエストに応答していることを確認しておくことが重要です。Elastic Beanstalk には、アプリケーションの応答性をモニタリングするのに役立つ機能が用意されています。アプリケーションに関する統計をモニタリングし、しきい値を超えたときにアラートを出します。

Elastic Beanstalk で提供される状態モニタリングの詳細については、「ベーシックヘルスレポート」を参照してください。

AWS Toolkit for Visual Studio または AWS マネジメントコンソールを使用して、アプリケーションに関する運用情報にアクセスできます。

Toolkit の [ステータス] フィールドでは環境のステータスとアプリケーションの状態が一目でわかります。

アプリケーションの状態を監視するには

1. AWS Toolkit for Visual Studio の AWS Explorer で、Elastic Beanstalk ノードを展開し、アプリケーションノードを展開します。
2. アプリケーション環境のコンテキスト (右クリック) メニューを開き、View Status (ステータスの表示) を選択します。
3. アプリケーション環境タブで、モニタリング を選択します。

   モニタリング パネルには、特定のアプリケーション環境に対するリソースの使用状況を示す一連のグラフが表示されます。

   > **Note**
   >
   > デフォルトでは、時間範囲は 1 時間前に設定されます。この設定を変更するには、時間範囲 リストで異なる時間範囲を選択します。

AWS Toolkit for Visual Studio または AWS マネジメントコンソールを使用して、アプリケーションに関連付けられたイベントを表示できます。

アプリケーションイベントを表示するには

1. AWS Toolkit for Visual Studio の AWS Explorer で、Elastic Beanstalk ノードとアプリケーションノードを展開します。
2. アプリケーション環境のコンテキスト (右クリック) メニューを開き、View Status (ステータスの表示) を選択します。
3. アプリケーション環境タブで イベント を選択します。

.NET on Windows Server プラットフォームから Elastic Beanstalk の .NET Core on Linux プラットフォームへの移行

.NET on Windows Server プラットフォームで実行されるアプリケーションを .NET Core on Linux プラットフォームに移行できます。Windows から Linux プラットフォームへの移行に関する考慮事項を次に示します。

.NET Core on Linux プラットフォームへの移行に関する考慮事項

[面積]	変更と情報
アプリケーションの設定	Windows プラットフォームでは、デプロイマニフェストを使用して、環境で実行されるアプリケーションを指定します。.NET Core on Linux プラットフォームは、Procfile を使用して、環境のインスタンスで実行されるアプリケーションを指定します。アプリケーションのバンドル化の詳細については、the section called “アプリケーションのバンドル”を参照してください。
プロキシサーバー	Windows プラットフォームでは、アプリケーションのプロキシサーバーとして IIS を使用します。.NET Core on Linux プラットフォームには、デフォルトでリバースプロキシとして nginx が含まれています。プロキシサーバーを使用せず、Kestrel をアプリケーションのウェブサーバとして使用することもできます。詳細については、「the section called “プロキシサーバー”」を参照してください。
ルーティング	Windows プラットフォームでは、アプリケーションコードで IIS を使用し、デプロイマニフェストを含めて IIS パスを設定します。.NET Core on Linux プラットフォームでは、アプリケーションコードで ASP .NET Core ルーティングを使用

[面積]	変更と情報
	して、お客様の環境の nginx 設定を更新します。詳細については、「the section called “プロキシサーバー”」を参照してください。
ログ	Linux プラットフォームと Windows プラットフォームは、異なるログをストリーミングします。詳細については、the section called “Elastic Beanstalk が CloudWatch Logs を設定する方法”を参照してください

Go アプリケーションを Elastic Beanstalk にデプロイする

この章では、Go ウェブアプリケーションを設定して AWS Elastic Beanstalkにデプロイする手順を説明します。Elastic Beanstalk を使用すると、Amazon Web Services を使用して簡単に Go ウェブアプリケーションのデプロイ、管理、スケーリングができます。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) または Elastic Beanstalk コンソールを使用すると、わずか数分でアプリケーションをデプロイできます。Elastic Beanstalk アプリケーションをデプロイした後も、EB CLI を使用してアプリケーションと環境を管理することも、Elastic Beanstalk コンソール AWS CLIまたは APIs を使用することもできます。

EB CLI を使用して Hello World Go ウェブアプリケーションを作成してデプロイするには、「Go の QuickStart」のステップバイステップの手順に従います。

トピック

- QuickStart: Elastic Beanstalk に Go アプリケーションをデプロイする
- Elastic Beanstalk 用の Go 開発環境の設定
- Elastic Beanstalk Go プラットフォームを使用する

QuickStart: Elastic Beanstalk に Go アプリケーションをデプロイする

この QuickStart チュートリアルでは、Go アプリケーションを作成し、 AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

> **Note**
>
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: Go アプリケーションを作成する

- ステップ 2: EB CLI を使用して Go アプリケーションをデプロイする
- ステップ 3: Elastic Beanstalk でアプリケーションを実行する
- ステップ 4: クリーンアップする
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

AWS アカウントをすでにお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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、通話呼び出し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するように求められます。

   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 Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 「 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

EB CLI

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

ステップ 1: Go アプリケーションを作成する

プロジェクトディレクトリを作成します。

```
~$ mkdir eb-go
~$ cd eb-go
```

次に、Elastic Beanstalk を使用してデプロイするアプリケーションを作成します。ここでは、"Hello World" という RESTful ウェブサービスを作成します。

この例では、サービスへのアクセスに使用されるパスに基づいて変更されるカスタマイズされた挨拶を出力します。

このディレクトリに以下のコンテンツで `application.go` という名前のテキスト・ファイルを作成します。

Example **`~/eb-go/application.go`**

```
package main

import (
 "fmt"
 "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
 if r.URL.Path == "/" {
  fmt.Fprintf(w, "Hello World! Append a name to the URL to say hello. For example, use
 %s/Mary to say hello to Mary.", r.Host)
 } else {
  fmt.Fprintf(w, "Hello, %s!", r.URL.Path[1:])
 }
}

func main() {
 http.HandleFunc("/", handler)
 http.ListenAndServe(":5000", nil)
}
```

ステップ 2: EB CLI を使用して Go アプリケーションをデプロイする

次に、アプリケーション環境を作成し、設定済みのアプリケーションを Elastic Beanstalk を使用してデプロイします。

環境を作成し、Go アプリケーションをデプロイするには

1. eb init コマンドを使用して EB CLI リポジトリを初期化します。

```
~/eb-go$ eb init -p go go-tutorial --region us-east-2
Application go-tutorial has been created.
```

このコマンドは、go-tutorial という名前のアプリケーションを作成し、ローカルリポジトリを設定して最新の Go プラットフォームバージョンで環境を作成します。

2. (オプション) eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH を使用して connect できるようにします。

```
~/eb-go$ eb init
Do you want to set up SSH for your instances?
(y/n): y
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従ってキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。

3. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。Elastic Beanstalk は、アプリケーションの zip ファイルを自動的にビルドし、ポート 5000 で起動します。

```
~/eb-go$ eb create go-env
```

Elastic Beanstalk が環境を作成するのに約 5 分かかります。

ステップ 3: Elastic Beanstalk でアプリケーションを実行する

環境を作成するプロセスが完了したら、eb open でウェブサイトを開きます。

```
~/eb-go$ eb open
```

お疲れ様でした。Elastic Beanstalk で Go アプリケーションをデプロイしました。これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ Window が開きます。

ステップ 4: クリーンアップする

アプリケーションでの作業が終了したら、環境を終了できます。Elastic Beanstalk は、環境に関連付けられているすべての AWS リソースを終了します。

EB CLI を使用して Elastic Beanstalk 環境を終了するには、次のコマンドを実行します。

```
~/eb-go$ eb terminate
```

AWS アプリケーションの リソース

1 つのインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありません。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。

- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

1 つ以上のサンプルアプリケーションをデプロイし、ローカルでアプリケーションを開発して実行する準備が整ったら、「Elastic Beanstalk 用の Go 開発環境の設定」を参照します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。

Elastic Beanstalk 用の Go 開発環境の設定

このトピックでは、アプリケーションを AWS Elastic Beanstalkにデプロイする前に、ローカルテストを行うように Go 開発環境を設定する手順について説明します。。また、便利なツールのインストール手順を提供するウェブサイトも参照します。

Go のインストール

Go アプリケーションをローカルで実行するには、Go をインストールします。特定のバージョンを必要としない場合は、Elastic Beanstalk でサポートされる最新バージョンを取得します。サポートされているバージョンの一覧については、AWS Elastic Beanstalk プラットフォームドキュメントの「Go」を参照してください。

Go は https://golang.org/doc/install からダウンロードできます。

AWS SDK for Go のインストール

アプリケーション内から AWS リソースを管理する必要がある場合は、次のコマンドを使用して AWS SDK for Go をインストールします。

```
$ go get github.com/aws/aws-sdk-go
```

詳細については、「AWS SDK for Go」を参照してください。

Elastic Beanstalk Go プラットフォームを使用する

このトピックでは、Elastic Beanstalk で Go アプリケーションを設定、ビルド、実行する方法について説明します。

AWS Elastic Beanstalk は、Go プログラミング言語のさまざまなバージョンの多数のプラットフォームブランチをサポートしています。完全なリストについては、「AWS Elastic Beanstalk プラットフォーム」ドキュメントの「Go」を参照してください。

シンプルな Go アプリケーションは、2 つの方法でアプリケーションをデプロイすることができます。

- アプリケーションのメインパッケージを含む `application.go` という名前のルートの出典ファイルに出典バンドルを提供します。Elastic Beanstalk は、次のコマンドを使用してバイナリを構築します。

  ```
  go build -o bin/application application.go
  ```

  アプリケーションが構築されると、Elastic Beanstalk はポート 5000 で起動します。
- [`application`] という名前のバイナリファイルに出典バンドルを提供します。バイナリファイルは、ソースバンドルのルートまたは出典バンドルの `bin/` ディレクトリにあります。`application` バイナリファイルを両方の場所に配置すると、Elastic Beanstalk は `bin/` ディレクトリのファイルを使用します。

  Elastic Beanstalk は、ポート 5000 でこのアプリケーションを起動します。

どちらの場合も、当社がサポートする Go プラットフォームブランチでは、`go.mod` というファイルでモジュール要件を指定することもできます。詳細については、Go ブログの Migrating to Go Modules を参照してください。

より複雑な Go アプリケーションは、2 つの方法でアプリケーションをデプロイすることができます。

- アプリケーションの出典ファイルを含む出典バンドルを、Buildfile および Procfile とともに設定します。Buildfile にはアプリケーションを構築するためのコマンドが含まれ、Procfile にはアプリケーションを実行するための指示が含まれます。
- アプリケーションのバイナリファイルを含む出典バンドルを、Procfile とともに設定します。Procfile には、アプリケーションを実行するための指示が含まれます。

Go プラットフォームには、静的アセットを提供し、トラフィックをアプリケーションに転送するためのプロキシサーバーが含まれています。アドバンストなシナリオでは、デフォルトのプロキシ設定を拡張または上書きできます。

Elastic Beanstalk Linux ベースのプラットフォームを拡張するさまざまな方法の詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

Go 環境の設定

Go プラットフォーム設定では、Amazon EC2 インスタンスの動作をうまくチューニングできます。Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を編集できます。

Elastic Beanstalk コンソールを使用して、Amazon S3 へのログローテーションを有効にでき、アプリケーションが環境から読むことができる変数を設定します。

Elastic Beanstalk コンソールで Go 環境を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

ログオプション

[Log Options (ログオプション)] セクションには、2 つの設定があります。

- インスタンスプロファイル – アプリケーションに関連付けられた Amazon S3 バケットへのアクセス許可が付与されているインスタンスプロファイルを指定します。

- [Enable log file rotation to Amazon S3] (Amazon S3 へのログファイルのローテーションの有効化) - アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーするかどうかを指定します。

静的ファイル

パフォーマンスを向上させるために、[Static files] (静的ファイル) セクションを使用して、ウェブアプリケーション内のディレクトリセットから静的ファイル (HTML、イメージなど) を配信するようにプロキシサーバーを設定することができます。ディレクトリごとに、仮想パスをディレクトリマッピングに設定します。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接　　処理します。

設定ファイルまたは、Elastic Beanstalk コンソールを使用した静的ファイルの設定の詳細については、「the section called “静的ファイル”」を参照してください。

環境プロパティ

環境プロパティ セクションでは、アプリケーションを実行している Amazon EC2 インスタンスの環境設定を指定できます。環境プロパティは、キーバリューのペアとしてアプリケーションに渡されます。

Elastic Beanstalk で実行される Go 環境内では、`os.Getenv` 関数を使用して環境変数にアクセスできます。たとえば、次のコードを使用して可変数に `API_ENDPOINT` という名前のプロパティを読み取ることができます。

```
endpoint := os.Getenv("API_ENDPOINT")
```

詳細については、「環境変数およびその他のソフトウェア設定」を参照してください。

Go 設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクをパフォーマンスできます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

Go プラットフォームは、プラットフォーム固有の名前空間を定義しません。`aws:elasticbeanstalk:environment:proxy:staticfiles` 名前空間を使用して、静的

ファイルを配信するようにプロキシを設定できます。詳細と例については、「the section called “静的ファイル”」を参照してください。

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

Amazon Linux AMI (Amazon Linux 2 より前の) Go プラットフォーム

Elastic Beanstalk Go 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションの追加情報を読んでください。

> **メモ**
>
> - このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
> - 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

Go Configuration ネームスペース — Amazon Linux AMI (AL1)

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクをパフォーマンスできます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

> **Note**
>
> このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。

Amazon Linux AMI Go プラットフォームは、すべてのプラットフォームでsupport される名前空間の他に、プラットフォーム固有の設定の名前空間を 1 つsupport します。名前空間 `aws:elasticbeanstalk:container:golang:staticfiles` により、ウェブアプリケーションのパスを、静的コンテンツをコンテナするアプリケーションの出典バンドルのフォルダにマッピングするオプションを定義できます。

たとえば、この設定ファイルは、プロキシサーバーに `staticimages` フォルダのファイルを、`/images` のパスで配信するように指示します。

Example .ebextensions/go-settings.config

```
option_settings:
  aws:elasticbeanstalk:container:golang:staticfiles:
    /html: statichtml
    /images: staticimages
```

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

Elastic Beanstalk で Procfile を使用したカスタムスタートコマンドの設定

Go アプリケーションをスタートするカスタムコマンドを指定するには、[`Procfile`] という名前のファイルを出典バンドルのルートに含めます。

`Procfile` 書き込みと使用の詳細については、「ビルドファイルと Procfile」を参照してください。

Example [Procfile]

```
web: bin/server
queue_process: bin/queue_processor
foo: bin/fooapp
```

メインアプリケーション `web` を呼び出し、`Procfile` の最初のコマンドとしてリストする必要があります。Elastic Beanstalk は、メインの `web` アプリケーションを環境のルート URL (例: `http://my-go-env.elasticbeanstalk.com`) で公開します。

Elastic Beanstalk は、名前に `web_` プレフィックスがないアプリケーションも実行しますが、これらのアプリケーションはインスタンス外から使用できません。

Elastic Beanstalk では、`Procfile` のプロセスは継続的に実行される必要があります。Elastic Beanstalk はこれらのアプリケーションをモニタリングし、終了されたアプリケーションをすべて再開します。短期間実行されるプロセスには、Buildfile コマンドを使用します。

(Amazon Linux 2 より前の) Amazon Linux AMI での Procfile の使用

Elastic Beanstalk Go 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションの追加情報を読んでください。

> **メモ**
>
> - このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
> - 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

ポートパッシング — Amazon Linux AMI (AL1)

> **Note**
>
> このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。

Elastic Beanstalk は nginx プロキシを設定し、アプリケーション用の `PORT` 環境プロパティで指定されたポート番号のアプリケーションにリクエストを転送します。アプリケーションは、このポートを常にリッスンする必要があります。アプリケーション内のこの可変数にアクセスするには、`os.Getenv("PORT")` メソッドを呼び出します。

Elastic Beanstalk は、`PORT` 内の最初のアプリケーションのポートの `Procfile` 環境プロパティで指定されたポート番号を使用します。`Procfile` 内の以降のアプリケーションでは、ポート番号は

100 ずつ増えていきます。PORT 環境プロパティが設定されていない場合、Elastic Beanstalk は初期ポートに 5000 を使用します。

前述の例では、PORT 環境プロパティは web アプリケーションで 5000、queue_process アプリケーションで 5100、foo アプリケーションで 5200 となります。

次のように、PORT オプションを [aws:elasticbeanstalk:application:environment] 名前空間で設定すると、最初のポートを指定できます。

```
option_settings:
  - namespace:  aws:elasticbeanstalk:application:environment
    option_name:  PORT
    value:  <first_port_number>
```

使用しているアプリケーションの環境プロパティの設定の詳細については、「オプション設定」を参照してください。

Elastic Beanstalk の Buildfile を使用したカスタムビルドと設定

Go アプリケーションのカスタム構築と設定コマンドを指定するには、[Buildfile] という名前のファイルを出典バンドルのルートに含めます。ファイル名では、大文字と小文字が区別されます。Buildfile には次の形式を使用します。

```
<process_name>: <command>
```

Buildfile 内のコマンドは、正規表現：^[A-Za-z0-9_]+:\s*.+$ に一致する必要があります。

Elastic Beanstalk は、Buildfile で実行されるアプリケーションをモニタリングしません。Buildfile は、短期間実行されてタスクが完了したら終了されるコマンドに使用します。長期間継続的に実行される必要のあるアプリケーションプロセスには、終了する代わりに Procfile を使用します。

次の Buildfile の例では、build.sh がシェルスクリプトとして、出典バンドルのルートに配置されています。

```
make: ./build.sh
```

Buildfile 内のすべてのパスは、出典バンドルのルートと関連します。前もってインスタンス上のファイルの場所がわかっている場合は、Buildfile に絶対パスを含めることができます。

プロキシサーバーを設定します

Elastic Beanstalk は、リバースプロキシとして nginx を使用し、ポート 80 の Elastic Load Balancing ロードバランサーにアプリケーションをマッピングします。Elastic Beanstalk では、デフォルトの nginx 設定が用意されています。これは拡張することも、独自の設定で完全に上書きすることもできます。

デフォルトでは、Elastic Beanstalk はポート 5000 でアプリケーションにリクエストを送信するように nginx プロキシを設定します。デフォルトのポートを上書きするには、`PORT` 環境プロパティを、主要なアプリケーションがリッスンするポートに設定します。

> **Note**
>
> アプリケーションがリッスンしているポートは、ロードバランサーからリクエストを受信するために nginx サーバーがリッスンするポートに影響を与えません。

ご使用のプラットフォームバージョンでプロキシサーバーを設定する

すべての AL2023/AL2 プラットフォームでは、統一されたプロキシ設定機能がサポートされています。AL2023/AL2 を実行中のプラットフォームバージョンでプロキシサーバーを設定する方法の詳細については、「リバースプロキシの設定」を参照してください。

(Amazon Linux 2 より前の) Amazon Linux AMI でのプロキシの設定

> **メモ**
>
> - このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
> - 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

Elastic Beanstalk Go 環境で Amazon Linux AMI プラットフォームバージョン (Amazon Linux 2 より前) を使用している場合は、このセクションの情報を読んでください。

デフォルトのプロキシ設定の拡張および上書き — Amazon Linux AMI (AL1)

Elastic Beanstalk はリバースプロキシとして nginx を使用し、ポート 80 のロードバランサーにアプリケーションをマッピングします。独自の nginx 設定を実行する場合は、ソースバンドルに `.ebextensions/nginx/nginx.conf` ファイルを含めることで、Elastic Beanstalk によるデフォルト設定を上書きすることができます。このファイルが存在する場合、Elastic Beanstalk は nginx 設定ファイルの代わりにこのファイルを使用します。

`nginx.conf` `http` ブロック内のディレクティブに加えて他のディレクティブを含める場合は、出典バンドルの `.ebextensions/nginx/conf.d/` ディレクトリに設定ファイルを追加することもできます。このディレクトリ内のすべてのファイルには、`.conf` 拡張子が必要です。

Elastic Beanstalk でのヘルスレポートとモニタリングの強化、自動アプリケーションマッピング、静的ファイルなどの Elastic Beanstalk の機能を活用するには、nginx 設定ファイルの `server` ブロックに次の行を含める必要があります。

```
include conf.d/elasticbeanstalk/*.conf;
```

Elastic Beanstalk への Java アプリケーションのデプロイ

この章では、Java アプリケーションを設定してデプロイする手順について説明します AWS Elastic Beanstalk。Elastic Beanstalk を使用すると、Amazon Web Services を使用して簡単に Java ウェブアプリケーションのデプロイ、管理、スケーリングができます。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) または Elastic Beanstalk コンソールを使用すると、わずか数分でアプリケーションをデプロイできます。Elastic Beanstalk アプリケーションをデプロイした後も、EB CLI を使用してアプリケーションと環境を管理することも、Elastic Beanstalk コンソール AWS CLIまたは APIs を使用することもできます。

EB CLI を使用して Hello World Java ウェブアプリケーションを作成してデプロイするには、「Java の QuickStart」のステップバイステップの手順に従います。EB CLI を使用して Tomcat ベースのプラットフォームにデプロイするシンプルな「Hello World」Java JSP アプリケーションを作成するステップバイステップの手順に関心がある場合は、「Tomcat での Java の QuickStart」を試してください。

Java プラットフォームブランチ

AWS Elastic Beanstalk は、Java アプリケーション用の 2 つのプラットフォームをサポートしています。

- Tomcat – Java サーブレットや JavaServer Pages (JSP) を使用して HTTP リクエストを処理するアプリケーション用のオープンソースのウェブコンテナである Apache Tomcat に基づくプラットフォーム。Tomcat は、マルチスレッド、宣言セキュリティの設定、および広範なカスタマイズを提供することにより、ウェブアプリケーション開発を容易にします。Elastic Beanstalk では、Tomcat の現在のメジャーバージョンごとにプラットフォームブランチが用意されています。詳細については、「Tomcat プラットフォーム」を参照してください。
- Java SE – ウェブコンテナを使用しないアプリケーションや、Jetty または GlassFish など Tomcat 以外を使用するアプリケーション用のプラットフォーム。アプリケーションで使用するライブラリの Java Archive (JAR) は、Elastic Beanstalk にデプロイするソースバンドルに含めることができます。詳細については、「Java SE プラットフォーム」を参照してください。

Tomcat と Java SE プラットフォームの両方の最新ブランチは Amazon Linux 2 以降に基づいており、Java SE AWS ディストリビューションである Corretto を使用します。これらのプラットフォームブランチの名前には、Java ではなく Corretto という単語が含まれています。

現在のプラットフォームバージョンのリストについては、AWS Elastic Beanstalk プラットフォームガイドの「Tomcat」および「Java SE」を参照してください。

AWS ツール

AWS には、Java および Elastic Beanstalk を操作するためのいくつかのツールが用意されています。選択したプラットフォームブランチに関係なく、AWS SDK for Java を使用して、Java アプリケーション内から他の AWS のサービスを使用できます。 AWS SDK for Java は、raw HTTP 呼び出しをゼロから記述することなく、アプリケーションコードから AWS APIs を使用できるようにするライブラリのセットです。

コマンドラインからアプリケーションを管理することを選択する場合は、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) をインストールし、これを使用して Elastic Beanstalk 環境を作成、モニタリング、管理します。アプリケーションの複数の環境を実行すると、EB CLI が Git と統合され、各環境と様々な Git ブランチを関連付けられるようになります。

トピック

- QuickStart: Elastic Beanstalk に Java アプリケーションをデプロイする
- QuickStart: Elastic Beanstalk に Tomcat 用の Java JSP ウェブアプリケーションをデプロイする
- Java 開発環境をセットアップする
- Java のその他の Elastic Beanstalk アプリケーションの例とチュートリアル
- Elastic Beanstalk Tomcat プラットフォームを使用する
- Elastic Beanstalk Java SE プラットフォームの使用
- Amazon RDS DB インスタンスを Java Elastic Beanstalk 環境に追加する
- Java ツールとリソース

QuickStart: Elastic Beanstalk に Java アプリケーションをデプロイする

この QuickStart チュートリアルでは、Java アプリケーションを作成し、 AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

> **Note**
>
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにはアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: Java アプリケーションを作成する
- ステップ 2: アプリケーションをローカルに実行する
- ステップ 3: EB CLI を使用して Java アプリケーションをデプロイする
- ステップ 4: Elastic Beanstalk でアプリケーションを実行する
- ステップ 5：クリーンアップ
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

AWS アカウントをすでにお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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、通話呼び出し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するように求められます。

   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、 のセキュリティを確保し AWS IAM Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 「 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

EB CLI

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

Java と Maven

ローカルマシンに Amazon Corretto がインストールされていない場合は、「Amazon Corretto ユーザーガイド」のインストール手順に従ってインストールできます。

次のコマンドを実行して、Java のインストールを確認します。

```
~$ java -version
```

このチュートリアルでは、Maven を使用します。Apache Maven Project ウェブサイトのダウンロードとインストールの手順に従います。Maven の詳細については、Apache Maven Project ウェブサイトの「Maven ユーザーセンター」を参照してください。

次のコマンドを実行して、Maven のインストールを確認します。

```
~$ mvn -v
```

ステップ 1: Java アプリケーションを作成する

プロジェクトディレクトリを作成します。

```
~$ mkdir eb-java
~$ cd eb-java
```

次に、Elastic Beanstalk を使用してデプロイするアプリケーションを作成します。ここでは、"Hello World" という RESTful ウェブサービスを作成します。

この例では、Spring Boot フレームワークを使用します。このアプリケーションは、ポート 5000 でリスナーを開きます。デフォルトでは、Elastic Beanstalk はポート 5000 でアプリケーションにリクエストを転送します。

次のファイルを作成します。

このファイルは、シンプルな Spring Boot アプリケーションを作成します。

Example **~/eb-java/src/main/java/com/example/Application.java**

```
package com.example;
```

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

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

このファイルは、ここで定義する文字列を返すマッピングを作成します。

Example **~/eb-java/src/main/java/com/example/Controller.java**

```
package com.example;

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

@RestController
public class Controller {

    @GetMapping("/")
    public String index() {
       return "Hello Elastic Beanstalk!";
    }
}
```

このファイルは、Maven プロジェクト設定を定義します。

Example **~/eb-java/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>3.2.3</version>
  </parent>

  <groupId>com.example</groupId>
  <artifactId>BeanstalkJavaExample</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <java.version>21</java.version>
  </properties>

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

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

</project>
```

このプロパティファイルは、デフォルトのポートを 5000 に上書きします。これは、Elastic Beanstalk が Java アプリケーションにトラフィックを送信するデフォルトのポートです。

Example **~/eb-java/application.properties**

```
server.port=5000
```

ステップ 2: アプリケーションをローカルに実行する

次のコマンドでアプリケーションをパッケージングします。

```
~/eb-java$ mvn clean package
```

次のコマンドでアプリケーションをローカルに実行します。

```
~/eb-java$ java -jar target/BeanstalkJavaExample-1.0-SNAPSHOT.jar
```

アプリケーションの実行中に、ブラウザで `http://127.0.0.1:5000/` に移動します。「Hello Elastic Beanstalk!」というテキストが表示されます。

ステップ 3: EB CLI を使用して Java アプリケーションをデプロイする

Java アプリケーションを Elastic Beanstalk にデプロイする前に、ビルドアプリケーションをディレクトリからクリーンアップし、Buildfile と Procfile を作成して、Elastic Beanstalk 環境でアプリケーションがビルドおよび実行される方法を制御します。

アプリケーションのデプロイを準備および設定するには

1. ビルドされたアプリケーションをクリーンアップします。

   ```
   ~/eb-java$ mvn clean
   ```

2. `Buildfile` を作成します。

   Example **`~/eb-java/Buildfile`**

   ```
   build: mvn clean package
   ```

   この `Buildfile` は、アプリケーションのビルドに使用されるコマンドを指定します。Java アプリケーション用の `Buildfile` を含めない場合、Elastic Beanstalk はアプリケーションのビルドを試みません。

3. `Procfile` を作成します。

   Example **`~/eb-java/Procfile`**

   ```
   web: java -jar target/BeanstalkJavaExample-1.0-SNAPSHOT.jar
   ```

   この `Procfile` は、アプリケーションの実行に使用されるコマンドを指定します。Java アプリケーション用の `Procfile` を含めない場合、Elastic Beanstalk はソースバンドルのルートに 1 つの JAR ファイルがあると想定し、`java -jar` コマンドを使用してそれを実行しようとします。

これで、アプリケーションをビルドして起動するための設定ファイルを設定できたので、デプロイする準備が整いました。

環境を作成し、Java アプリケーションをデプロイするには

1. eb init コマンドを使用して EB CLI リポジトリを初期化します。

```
~/eb-java eb init -p corretto java-tutorial --region us-east-2

Application java-tutorial has been created.
```

   このコマンドは、`java-tutorial` という名前のアプリケーションを作成し、ローカルリポジトリを設定して最新の Java プラットフォームバージョンで環境を作成します。

2. (オプション) eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH を使用して connect できるようにします。

```
~/eb-java$ eb init
Do you want to set up SSH for your instances?
(y/n): y
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

   1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従ってキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。

3. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。Elastic Beanstalk は、アプリケーションの zip ファイルを自動的にビルドし、ポート 5000 で起動します。

```
~/eb-java$ eb create java-env
```

   Elastic Beanstalk が環境を作成するのに約 5 分かかります。

ステップ 4: Elastic Beanstalk でアプリケーションを実行する

環境を作成するプロセスが完了したら、eb open でウェブサイトを開きます。

```
~/eb-java eb open
```

お疲れ様でした。Elastic Beanstalk で Java アプリケーションをデプロイしました。これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ Window が開きます。

ステップ 5：クリーンアップ

アプリケーションでの作業が終了したら、環境を終了できます。Elastic Beanstalk は、環境に関連付けられているすべての AWS リソースを終了します。

EB CLI を使用して Elastic Beanstalk 環境を終了するには、次のコマンドを実行します。

```
~/eb-java$ eb terminate
```

AWS アプリケーションの リソース

1 つのインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありません。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。

- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

> **その他のチュートリアルを試す**
>
> 異なるアプリケーション例の他のチュートリアルを試したい場合は、「サンプルアプリケーションとチュートリアル」を参照してください。

1 つか 2 つのサンプルアプリケーションをデプロイし、ローカルにアプリケーションを開発して実行する準備が整ったら、「Java 開発環境をセットアップする」を参照します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。

QuickStart: Elastic Beanstalk に Tomcat 用の Java JSP ウェブアプリケーションをデプロイする

このチュートリアルでは、JavaServer Page (JSP) を使用してシンプルな Java ウェブアプリケーションを作成するプロセスについて説明します。1 つの Elastic Beanstalk 環境に複数のウェブアプリケーションを WAR ファイル形式でバンドルする場合は、「Tomcat 環境用に複数の WAR ファイルをバンドルする」を参照してください。

> **Note**
>
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: Java JSP アプリケーションを作成する
- ステップ 2: EB CLI を使用して Java JSP アプリケーションをデプロイする
- ステップ 3: Elastic Beanstalk でアプリケーションを実行する
- ステップ 4: クリーンアップする
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

AWS アカウントをすでにお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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、通話呼び出し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するように求められます。

   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 Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 「 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

EB CLI

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

ステップ 1: Java JSP アプリケーションを作成する

プロジェクトディレクトリを作成します。

```
~$ mkdir eb-tomcat
~$ cd eb-tomcat
```

次に、Elastic Beanstalk を使用してデプロイするアプリケーションを作成します。「Hello World」ウェブアプリケーションを作成します。

`index.jsp` という名前のシンプルな JSP ファイルを作成します。

Example **`~/eb-tomcat/index.jsp`**

```
<html>
  <body>
    <%out.println("Hello Elastic Beanstalk!");%>
  </body>
</html>
```

ステップ 2: EB CLI を使用して Java JSP アプリケーションをデプロイする

次のコマンドを実行して、このアプリケーションの Elastic Beanstalk 環境を作成します。

環境を作成し、Java JSP アプリケーションをデプロイするには

1. eb init コマンドを使用して EB CLI リポジトリを初期化します。

```
~/eb-tomcat$ eb init -p tomcat tomcat-tutorial --region us-east-2
```

   このコマンドは、`tomcat-tutorial` という名前のアプリケーションを作成し、ローカルリポジトリを設定して最新の Tomcat プラットフォームバージョンで環境を作成します。

2. (オプション) eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH を使用して connect できるようにします。

```
~/eb-go$ eb init
Do you want to set up SSH for your instances?
(y/n): y
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

   1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従ってキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。

3. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。Elastic Beanstalk は、アプリケーションの zip ファイルを自動的にビルドし、ポート 5000 で起動します。

```
~/eb-tomcat$ eb create tomcat-env
```

   Elastic Beanstalk が環境を作成するのに約 5 分かかります。

ステップ 3: Elastic Beanstalk でアプリケーションを実行する

環境を作成するプロセスが完了したら、eb open でウェブサイトを開きます。

```
~/eb-tomcat$ eb open
```

お疲れ様でした。Elastic Beanstalk で Java JSP アプリケーションをデプロイしました。これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ Window が開きます。

ステップ 4: クリーンアップする

アプリケーションでの作業が終了したら、環境を終了できます。Elastic Beanstalk は、環境に関連付けられているすべての AWS リソースを終了します。

EB CLI を使用して Elastic Beanstalk 環境を終了するには、次のコマンドを実行します。

```
~/eb-tomcat$ eb terminate
```

AWS アプリケーションの リソース

1 つのインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありません。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。

- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

> **その他のチュートリアルを試す**
>
> 異なるアプリケーション例の他のチュートリアルを試したい場合は、「サンプルアプリケーションとチュートリアル」を参照してください。

1 つか 2 つのサンプルアプリケーションをデプロイし、ローカルの Tomcat ウェブコンテナで Java アプリケーションを開発して実行する準備が整ったら、「Java 開発環境をセットアップする」を参照します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。

Java 開発環境をセットアップする

このトピックでは、Java 開発環境を設定し、アプリケーションを AWS Elastic Beanstalkにデプロイする前にローカルでテストする手順について説明します。また、便利なツールのインストール手順を提供するウェブサイトも参照します。

セクション

- Java 開発キットをインストールする
- ウェブ コンテナをインストールする
- ライブラリをダウンロードする
- AWS SDK for Java のインストール
- IDE またはテキストエディタをインストールする

Java 開発キットをインストールする

Java 開発キット (JDK) をインストールする 指定しない場合は、最新バージョンを取得します。oracle.com で JDK をダウンロードします。

JDK には、ソースファイルを Elastic Beanstalk ウェブサーバーで実行可能なクラスファイルに組み込む目的で使用できる Java コンパイラが含まれています。

ウェブ コンテナをインストールする

まだ別のウェブコンテナまたはフレームワークがない場合は、Elastic Beanstalk が Amazon Linux オペレーティングシステムでサポートするバージョンの Tomcat をインストールします。Elastic Beanstalk が現在サポートする Apache Tomcat のバージョンのリストについては、「AWS Elastic Beanstalk プラットフォーム」ドキュメントの「Tomcat」を参照してください。Apache Tomcat ウェブサイトから、環境に適用される Tomcat バージョンをダウンロードします。

ライブラリをダウンロードする

Elastic Beanstalk プラットフォームには、デフォルトでいくつかのライブラリが含まれています。アプリケーションが使用するライブラリをダウンロードしてプロジェクトフォルダに保存し、アプリケーションソースバンドルにデプロイします。

Tomcat をローカルでインストールしている場合、インストール フォルダからサーブレット API および JavaServer Pages (JSP) API ライブラリをコピーできます。Tomcat プラットフォームの

バージョンをデプロイする場合は、これらのファイルをソースバンドルに含める必要はありませんが、`classpath` に組み込んでこれらを使用するすべてのクラスをコンパイルする必要があります。

JUnit、Google Guava、Apache Commons は、複数の便利なライブラリを提供します。詳細についてはそれぞれのホームページにアクセスしてください。

- JUnit をダウンロードする
- Google Guava をダウンロードする
- Apache Commons をダウンロードする

AWS SDK for Java のインストール

アプリケーション内から AWS リソースを管理する必要がある場合は、 AWS SDK for Java をインストールします。たとえば、 では AWS SDK for Java、Amazon DynamoDB (DynamoDB) を使用して、複数のウェブサーバー間で Apache Tomcat アプリケーションのセッション状態を共有できます。詳細については、 AWS SDK for Java ドキュメントの「Manage Tomcat Session State with Amazon DynamoDB」を参照してください。

詳細とインストール方法については、AWS SDK for Java のホームページにアクセスしてください。

IDE またはテキストエディタをインストールする

統合された開発環境（IDE）は、アプリケーション開発を用意にする幅広い機能を提供します。Java 開発用の IDE を使用していない場合は、Eclipse と IntelliJ を試してどちらが使いやすいかを確認してください。

- Eclipse IDE for Java EE Developers をインストールする
- IntelliJ をインストールする

IDE では、ソースコントロールにコミットする必要がないファイルがプロジェクトフォルダに追加される場合があります。ソースコントロールにこれらのファイルがコミットされないようにするには、`.gitignore` または同等のソースコントロールツールを使用します。

IDE の特徴のすべては必要なく、単純にコーディングを開始する場合は、Sublime Text のインストールを検討します。

> **Note**
>
> 2023 年 5 月 31 日に AWS Toolkit for Eclipse のサポートが終了したため、 AWSによるサポートも終了しました。のライフサイクルの終了の詳細については AWS Toolkit for Eclipse、 AWS Toolkit for Eclipse GitHub リポジトリの README.md ファイルを参照してください。

Java のその他の Elastic Beanstalk アプリケーションの例とチュートリアル

このセクションでは、その他のアプリケーションとチュートリアルについて説明します。このトピックで前述した Java の QuickStart および Tomcat での Java の QuickStart トピックでは、EB CLI を使用したサンプル Java アプリケーションの起動について説明します。

で Java アプリケーションの使用を開始するには AWS Elastic Beanstalk、最初のアプリケーションバージョンとしてアップロードし、 環境にデプロイするアプリケーションソースバンドルが必要です。環境を作成すると、スケーラブルなウェブアプリケーションを実行するために必要なすべての AWS リソースが Elastic Beanstalk によって割り当てられます。

サンプル Java アプリケーションで環境を起動する

Elastic Beanstalk は、各プラットフォームの単一ページのサンプルアプリケーションと、Amazon RDS や言語、プラットフォーム固有の機能や APIs などの追加 AWS リソースの使用を示すより複雑な例を提供します。

単一ページのサンプルは、環境を作成するときに取得する同じコードであり、独自のソースコードを提供する必要はありません。複雑なサンプルアプリケーションは GitHub でホストされていますが、Elastic Beanstalk 環境にデプロイする前にはコンパイルやビルドが必要になる場合があります。

サンプル

名前	サポートされるバージョン	環境タイプ	送信元	説明
Tomc (単一ページ)	すべての Tomcat with Corretto プラットフォームブランチ	ウェブサーバー ワーカー	tomcat.zip	1 ページ (`index.jsp`) がウェブサイトのルートに表示されるように設定された Tomcat ウェブアプリケーションです。 ワーカー環境では、このサンプルには、1 分に 1 回 [`cron.yaml`] を呼び出すスケジュール済みタスク設定する [`scheduled.jsp`] ファイルが含まれます。`scheduled.jsp` が呼び出されると、`/tmp/sample-app.log` のログファイルに書き込まれます。最後に、設定ファイルが `.ebextensions` に含められます。この設定ファイルにより、`/tmp/` のログは、環境ログがリクエストされたときに Elastic Beanstalk が読み取る場所にコピーされます。 このサンプルを実行している環境で X-Ray の統合を有効にすると、アプリケーションは X-Ray に関する追加のコンテンツを表示し、X-Ray コンソールで表示できるデバッグ情報を生成するオプションを提供します。
Corre (単一ページ)	Corretto 11 Corretto 8	ウェブサーバー	corretto.zip	`Buildfile` 設定ファイルおよび `Procfile` 設定ファイルを使用する Corretto アプリケーションです。 このサンプルを実行している環境で X-Ray の統合を有効にすると、アプリケーション

名前	サポートされるバージョン	環境タイプ	送信元	説明
				は X-Ray に関する追加のコンテンツを表示し、X-Ray コンソールで表示できるデバッグ情報を生成するオプションを提供します。

名前	サポートされるバージョン	環境タイプ	送信元	説明
Score	Java 8	ウェブサーバー	クローン the repo at GitHub.com	Scorekeep は、Spring フレームワークを使ってユーザー、セッション、およびゲームを作成および管理するためのインターフェイスを提供する、RESTful ウェブ API です。API は、HTTP を介して API を使用する Angular 1.5 ウェブアプリを持つバンドルです。 アプリケーションは、Java SE プラットフォームの機能を使って依存関係をダウンロードし、オンインスタンスを構築することで、ソースバンドルのサイズを最小化します。また、アプリケーションには、プロキシを通じてポート 80 で静的にフロントエンドウェブアプリに対応するデフォルト設定を上書きし、`/api` で実行される API に `localhost:5000` 以下のパスをルーティングする nginx 設定ファイルが含まれています。 Scorekeep には、`xray` で使用する Java アプリケーションの設定方法を示す AWS X-Rayブランチも含まれます。サーブレットフィルターを使用した受信 HTTP リクエストの計測、自動および手動 AWS SDK クライアントの計測、レコーダー設定、送信 HTTP リクエストと SQL クライアントの計測を示します。 手順については readme を参照するか、AWS X-Ray 入門チュートリアルを使

名前	サポートされるバージョン	環境タイプ	送信元	説明
				用して X-Ray でアプリケーションをでお試しください。
Does it Have Snak	Tomcat 8 と Java 8	ウェブサーバー	クローン the repo at GitHub.com	Does it Have Snakes? は、Elastic Beanstalk 設定ファイル、Amazon RDS、JDBC、PostgreSQL、サーブレット、JSP、簡易タグのサポート、タグファイル、Log4J、ブートストラップ、Jackson の使用方法を示す Tomcat ウェブアプリケーションです。 このプロジェクトのソースコードには、クラスファイルにサーブレットとモデルをコンパイルし、Elastic Beanstalk 環境にデプロイできるウェブアーカイブに必要ファイルをパッケージする、最小限のビルドスクリプトが含まれています。完全な手順については、プロジェクトのリポジトリ内の readme ファイルを参照してください。
Locu Load Gene	Java 8	ウェブサーバー	クローン the repo at GitHub.com	別の Elastic Beanstalk 環境で実行している別のウェブアプリケーションの負荷テストに使用できるウェブアプリケーションです。`Buildfile` ファイル、`Procfile` ファイル、DynamoDB、Locust 、オープンソースの負荷テストツールの使用方法を示しています。

サンプルアプリケーションをダウンロードし、以下の手順に従って Elastic Beanstalk にデプロイします。

サンプルアプリケーションのある環境を起動する場合 (コンソール)

1. Elastic Beanstalk 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、[アプリケーション] を選択します。リストから既存のアプリケーションを選択します。 アプリケーションの管理 の手順に従って作成することもできます。
3. アプリケーションの概要ページで、[Create new environment] (新しい環境の作成) を選択します。

   次の図は、アプリケーションの概要ページを示しています。

   これにより、[Create environment] (環境を作成する) ウィザードが起動します。ウィザードには、新しい環境を作成するための一連のステップが用意されています。
4. [環境枠] では、[ウェブサーバー環境] または [ワーカー環境] の環境枠を選択します。作成後に環境枠を変更することはできません。

   > **Note**
   > Windows Server プラットフォームの .NET はワーカー環境枠をsupport していません。

   [アプリケーション情報] フィールドは、以前に選択したアプリケーションに基づいてデフォルトで設定されます。

   [環境情報] では、アプリケーション名に基づいて [環境名] がデフォルトでグループ化されます。別の環境名を使用する場合は、フィールドに別の値を入力できます。必要に応じて [ドメイン] の名前を入力できます。入力しない場合、Elastic Beanstalk は値を自動的に生成します。必要に応じて [環境の説明] を入力することもできます。
5. プラットフォーム では、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。

   > **Note**
   > Elastic Beanstalk では、一覧表示されるほとんどのプラットフォームで複数のバージョンがサポートされています。デフォルトでは、選択したプラットフォームとプラットフォームブランチの推奨バージョンがコンソールによって選択されます。アプリケーションで異なるバージョンが必要な場合は、ここでそのバージョンを選択できます。サ

ポートされているプラットフォームのバージョンについては、the section called “サポートされているプラットフォーム” を参照してください。

6. [アプリケーションコード] では、サンプルアプリケーションを起動するためのいくつかの選択肢があります。
   - ソースコードを指定せずにデフォルトのサンプルアプリケーションを起動するには、[サンプルアプリケーション] を選択します。このアクションは、以前に選択したプラットフォームに対して Elastic Beanstalk が提供する単一ページアプリケーションを選択します。
   - このガイドまたは他のソースからサンプルアプリケーションをダウンロードした場合は、次の手順を実行します。
     a. [コードのアップロード] を選択します。
     b. 次に [ローカルファイル] を選択し、[アプリケーションをアップロード] で [ファイルを選択] を選択します。
     c. コンピュータのオペレーティングシステムには、ダウンロードしたローカルファイルを選択するためのインターフェイスが表示されます。ソースバンドルファイルを選択して続行します。
7. [プリセット] では、[単一インスタンス] を選択します。
8. [次へ] を選択します。
9. [サービスアクセスの設定] ページが表示されます。

   次の図は、[サービスアクセスの設定] ページを示しています。

10. [既存のサービスロール] ドロップダウンから値を選択します。
11. (オプション) EC2 キーペアを以前に作成している場合は、[EC2 キーペア] フィールドドロップダウンから選択できます。これを使用して、Elastic Beanstalk がアプリケーション用にプロビジョニングした Amazon EC2 インスタンスに安全にログインできます。このステップをスキップした場合、環境の作成後に EC2 キーペアをいつでも作成して割り当てることができます。詳細については、「EC2 キーペア」を参照してください。
12. 次に、[EC2 インスタンスプロファイル] ドロップダウンリストに焦点を当てます。このドロップダウンリストに表示される値は、アカウントが以前に新しい環境を作成したかどうかによって異なる場合があります。

    リストに表示されている値に基づいて、次のいずれかの項目を選択します。

- `aws-elasticbeanstalk-ec2-role` がドロップダウンリストに表示されている場合は、それをドロップダウンリストから選択します。
- リストに別の値が表示され、かつそれが環境向けのデフォルト EC2 インスタンスプロファイルである場合、その値をドロップダウンリストから選択します。
- [EC2 インスタンスプロファイル] ドロップダウンリストに値が表示されない場合は、インスタンスプロファイルを作成する必要があります。

> **インスタンスプロファイルを作成する**
>
> インスタンスプロファイルを作成するには、同じページの別の手順に迂回します。この手順の末尾に移動し、次の手順である「EC2 インスタンスプロファイル の IAM ロールを作成する」を展開します。
> 「EC2 インスタンスプロファイルの IAM ロールを作成する」ステップを完了し、その後 [EC2 インスタンスプロファイル] 用に選択できる IAM ロールを作成します。その後、このステップに戻ります。

IAM ロールを作成してリストを更新すると、ドロップダウンリストに選択肢として表示されます。[EC2 インスタンスプロファイル] ドロップダウンリストから、先ほど作成した IAM ロールを選択します。

13. [Configure service access] (サービスアクセスの設定) ページで [Skip to Review] (確認をスキップ) を選択します。

    これにより、このステップのデフォルト値が選択され、オプションのステップはスキップされます。

14. [Review] (レビュー) ページに、すべての選択内容の概要が表示されます。

    環境をさらにカスタマイズするには、設定する項目を含むステップの横にある [Edit] (編集) を選択します。以下のオプションは、環境の作成中にのみ設定できます。

    - 環境名
    - ドメイン名
    - プラットフォームのバージョニング
    - プロセッサ
    - VPC

- 階層

次の設定は環境の作成後に変更できますが、新しいインスタンスあるいはその他のリソースをプロビジョニングする必要があり、適用までに長い時間がかかる場合があります。

- インスタンスタイプ、ルートボリューム、キーペア、および AWS Identity and Access Management (IAM) ロール
- 内部 Amazon RDS データベース
- ロードバランサー

すべての使用できる設定の詳細については、「新しい環境の作成ウィザード」を参照してください。

15. ページ下部の [Submit] (送信) を選択して、新しい環境の作成を開始します。

EC2 インスタンスプロファイルの IAM ロールを作成

EC2 インスタンスプロファイルを作成するには

1. [ロールの作成] を選択してください。
2. [信頼できるエンティティタイプ] で、[AWS サービス] を選択します。
3. ユースケースでは、Elastic Beanstalk – Compute を選択します。
4. [次へ] を選択します。
5. アクセス許可ポリシーに以下が含まれていることを確認し、次へを選択します。
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`
6. [ロールの作成] を選択してください。
7. サービスアクセスの設定タブに戻り、リストを更新して、新しく作成した EC2 インスタンスプロファイルを選択します。

次のステップ

環境でアプリケーションを実行すると、アプリケーションの新しいバージョンや、まったく異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。

サンプルアプリケーションを 1 つか 2 つデプロイし、Java アプリケーションをローカルで開発および実行する準備が整ったら、次のセクションを参照して、必要なすべてのツールやライブラリとともに Java 開発環境を設定します。

Elastic Beanstalk Tomcat プラットフォームを使用する

このトピックでは、Elastic Beanstalk Tomcat プラットフォームで実行される Java アプリケーションを設定、ビルド、実行する方法について説明します。

AWS Elastic Beanstalk Tomcat プラットフォームは、Tomcat ウェブコンテナで実行できる Java ウェブアプリケーションのプラットフォームバージョンのセットです。Tomcat は、nginx プロキシサーバーの背後で実行されます。各プラットフォームブランチは Tomcat の 1 つのメジャーバージョンに対応しています。

設定オプションは実行中の環境の設定を変更するために Elastic Beanstalk コンソールで利用できます。環境を終了したときにその設定が失われないようにするため、保存済み設定を使用して設定を保存し、それを後で他の環境に適用することができます。

ソースコードの設定を保存する場合、設定ファイルを含めることができます。設定ファイルの設定は、環境を作成するたびに、またはアプリケーションをデプロイするたびに適用されます。設定ファイルを使用して、デプロイの間にパッケージをインストールしたり、スクリプトを実行したり、他のインスタンスのカスタマイズオペレーションを実行することもできます。

Elastic Beanstalk Tomcat プラットフォームには、アプリケーションにリクエストを転送するリバースプロキシが含まれています。アプリケーションの負荷を減らすため、ソースコードのフォルダーから静的アセットに対応するようプロキシサーバーを設定する設定オプションを使用できます。高度なシナリオでは、ソースバンドルに独自の `.conf` ファイルを含めて Elastic Beanstalk のプロキシ設定を拡張するか、これを完全に上書きできます。

> **Note**
>
> Elastic Beanstalk は、Tomcat プラットフォームのプロキシサーバーとして nginx (デフォルト) および Apache HTTP サーバーをサポートします。Elastic Beanstalk Tomcat 環境で

(Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームブランチを使用している場合は、Apache HTTP Server Version 2.2 を使用することもできます。これらの古いプラットフォームブランチでは、Apache (最新) がデフォルトです。
2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

ウェブアプリケーションアーカイブ (WAR) ファイルの Java アプリケーションは固有の構造でパッケージングする必要があります。必要な構造と、その構造をプロジェクトディレクトリの構造に関連付ける方法については、「プロジェクトフォルダーを構築する」を参照してください。

同じウェブサーバーで複数のアプリケーションサーバーを実行するため、単一のソースバンドルに複数の WAR ファイルをバンドルすることができます。複数のソースバンドルの各アプリケーションは、ルートパス (`ROOT.war` で実行される [*myapp*`.elasticbeanstalk.com/`]) またはその下のパスディレクトリ (`app2.war` で実行される [*myapp*`.elasticbeanstalk.com/`*app2*`/`.war]) のいずれかで実行されます (どちらかは WAR の名前によって決まります)。単一の WAR ソースバンドルでは、アプリケーションは常にルートパスで実行されます。

Elastic Beanstalk コンソールで適用される設定は、設定ファイルに同じ設定があれば、それらの設定を上書きします。これにより、設定ファイルでデフォルト設定を定義し、コンソールでそのデフォルト設定を環境固有の設定で上書きできます。設定の優先順位の詳細と設定の他の変更方法については、「設定オプション」を参照してください。

Elastic Beanstalk Linux ベースのプラットフォームを拡張するさまざまな方法の詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

トピック

- Tomcat 環境を設定する
- Tomcat 設定の名前空間
- Tomcat 環境用に複数の WAR ファイルをバンドルする
- プロジェクトフォルダーを構築する
- プロキシサーバーを設定します

Tomcat 環境を設定する

Elastic Beanstalk Tomcat プラットフォームには、すべてのプラットフォームに用意されている標準オプションに加えて、プラットフォーム固有のオプションがいくつかあります。これらのオプションにより、環境のウェブサーバーで実行される Java 仮想マシン (JVM) を設定し、アプリケーションに情報設定文字列を提供するシステムプロパティを定義できます。

Elastic Beanstalk コンソールを使用して、Amazon S3 へのログローテーションを有効にし、アプリケーションが環境から読むことができる変数を設定することができます。

Elastic Beanstalk コンソールで Tomcat 環境を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

コンテナオプション

次のプラットフォーム固有のオプションを指定できます。

- [プロキシサーバー] – 環境インスタンスで使用するプロキシサーバーです。デフォルトでは、nginx が使用されます。

JVM コンテナオプション

Java 仮想マシン (JVM) のヒープサイズは、ガベージコレクションが発生する前にアプリケーションがメモリで作成できるオブジェクトの数を決定します。JVM の初期ヒープ サイズ (`-Xms option`) と JVM の最大ヒープ サイズ (`-Xmx` オプション) を変更できます。初期ヒープサイズを大きく設定すると、ガベージコレクションの発生前により多くのオブジェクトを作成できますが、ガベージコレクタがヒープを圧縮する時間が長くなります。最大ヒープサイズは、多量の作業を実行中にヒープを拡張する場合に JVM が割り当てることのできる最大メモリ容量を指定します。

> **Note**
>
> 使用可能なメモリは、Amazon EC2 インスタンスタイプによって異なります。Elastic Beanstalk 環境で使用可能な EC2 インスタンスタイプの詳細については、Amazon Elastic

Compute Cloud Linux インスタンス用ユーザーガイド の「インスタンスタイプ」を参照してください。

JVM ヒープの永続世代は、クラス定義と関連メタデータを保存するセクションです。永続世代のサイズを変更するには、[JVM PermGen の最大サイズ] (`-XX:MaxPermSize`) オプションに新しいサイズを入力します。この設定が適用されるのは、Java 7 以前のみです。このオプションは JDK 8 で廃止され、[MaxMetaspace サイズ] (`-XX:MaxMetaspaceSize`) オプションに置き換えられました。

> **Important**
>
> JDK 17 では、Java `-XX:MaxPermSize` オプションのサポートが削除されました。Corretto 17 を搭載した Elastic Beanstalk プラットフォームブランチで実行されている環境でこのオプションを使用すると、エラーが発生します。Elastic Beanstalk は、2023 年 7 月 13 日に Corretto 17 を搭載した Tomcat を実行する最初のプラットフォームブランチをリリースしました。
> 詳細については、以下のリソースを参照してください。
>
> - Oracle Java ドキュメンテーション Web サイト: Java オプションが削除されました
> - Oracle Java ドキュメントの Web サイト:「その他の考慮事項」の「クラスメタデータ」セクション

Elastic Beanstalk プラットフォームとそのコンポーネントの詳細については、AWS Elastic Beanstalk プラットフォームガイドの「サポートされているプラットフォーム」を参照してください。

ログオプション

[Log Options] セクションには、2 つの設定があります。

- インスタンスプロファイル – アプリケーションに関連付けられた Amazon S3 バケットへのアクセス許可が付与されているインスタンスプロファイルを指定します。
- [Enable log file rotation to Amazon S3] (Amazon S3 へのログファイルのローテーションの有効化) - アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーするかどうかを指定します。

静的ファイル

パフォーマンスを向上させるために、[Static files] (静的ファイル) セクションを使用して、ウェブアプリケーション内のディレクトリセットから静的ファイル (HTML、イメージなど) を配信するようにプロキシサーバーを設定することができます。ディレクトリごとに、仮想パスをディレクトリマッピングに設定します。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接　　処理します。

設定ファイルまたは、Elastic Beanstalk コンソールを使用した静的ファイルの設定の詳細については、「the section called “静的ファイル”」を参照してください。

環境プロパティ

[環境プロパティ] セクションでは、アプリケーションを実行している Amazon EC2 インスタンスの環境設定を指定できます。環境プロパティは、キーと値のペアでアプリケーションに渡されます。

Tomcat プラットフォームは、必要に応じて外部データベースに接続文字列を渡すため、`JDBC_CONNECTION_STRING` という名前の Tomcat 環境のプレースホルダプロパティを定義します。

> **Note**
>
> RDS DB インスタンスを環境にアタッチする場合は、Elastic Beanstalk によって提供される Amazon Relational Database Service (Amazon RDS) 環境プロパティから JDBC 接続文字列を動的に構築します。JDBC_CONNECTION_STRING は、Elastic Beanstalk でプロビジョニングされないデータベースインスタンスにのみ使用します。
> Java アプリケーションで Amazon RDS を使用する方法の詳細については、「」を参照してくださいAmazon RDS DB インスタンスを Java Elastic Beanstalk 環境に追加する

2025 年 3 月 26 日より前にリリースされた Tomcat プラットフォームバージョンでは、 を使用して環境変数にアクセスできます`System.getProperty()`。たとえば、次のコードを持つ変数`API_ENDPOINT`から という名前のプロパティを読み取ることができます。

```
String endpoint = System.getProperty("API_ENDPOINT");
```

2025 年 3 月 26 日以降にリリースされた Tomcat プラットフォームバージョンでは、`System.getenv`を使用してプレーンテキストの環境変数にアクセスすることもできます。

引き続き System.getPropertyを使用して、プレーンテキストの環境変数にアクセスできます。ただし、シークレットとして保存されている環境変数は、 を使用してのみ使用できますSystem.getenv。たとえば、次のコードAPI_KEYを使用して という名前の環境変数を読み取ることができます。

```
String apiKey = System.getenv("API_KEY");
```

> **Important**
>
> 2025 年 3 月 26 日以降にリリースされた Tomcat プラットフォームバージョンの環境変数System.getenv()へのアクセスを追加すると、Java システムプロパティよりも環境変数を優先するアプリケーションや、 から System.getProperty()に明示的に切り替えるアプリケーションで予期しない動作が発生する可能性がありますSystem.getenv()。 https://docs.aws.amazon.com/elasticbeanstalk/latest/relnotes/release-2025-03-26-windows.html
> システムプロパティ (コマンドラインを介して渡される) では特殊文字のシェル エスケープが必要であるのに対し、環境変数はそうではないため、Java システムプロパティの代わりに環境変数を使用すると、値は異なる方法で解決される場合があります。
> アプリケーションが影響を受ける場合は、次の点を考慮してください。
>
> - の使用時に環境プロパティ値からエスケープ文字を削除する System.getenv()
> - を明示的に使用するようにアプリケーションを設定する System.getProperty()
> - アップグレード時にアプリケーションを徹底的にテストして、一貫した動作を確保する

詳細については、「環境変数およびその他のソフトウェア設定」を参照してください。

Tomcat 設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクを実行できます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

Tomcat プラットフォームでは、すべての Elastic Beanstalk 環境でサポートされるオプションに加えて、以下の名前空間のオプションがサポートされます。

- aws:elasticbeanstalk:container:tomcat:jvmoptions – JVM 設定を変更します。この名前空間のオプションは、次のように管理コンソールのオプションに対応します。

- `Xms` – JVM コマンドラインオプション
- `JVM Options` – JVM コマンドラインオプション

- `aws:elasticbeanstalk:environment:proxy` – 環境のプロキシサーバーを選択します。

次の例の設定ファイルは、Tomcat 固有の設定オプションの使用を示しています。

Example .ebextensions/tomcat-settings.config

```
option_settings:
  aws:elasticbeanstalk:container:tomcat:jvmoptions:
    Xms: 512m
    JVM Options: '-Xmn128m'
  aws:elasticbeanstalk:application:environment:
    API_ENDPOINT: mywebapi.zkpexsjtmd.us-west-2.elasticbeanstalk.com
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
```

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については、「設定オプション」を参照してください。

Amazon Linux AMI (Amazon Linux 2 以前の) Tomcat プラットフォーム

Elastic Beanstalk Tomcat 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションの追加情報をお読みください。

> **メモ**
>
> - このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
> - 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

Tomcat 設定ネームスペース — Amazon Linux AMI (AL1)

Tomcat Amazon Linux AMI プラットフォームは、次の名前空間で追加オプションをサポートしています。

- `aws:elasticbeanstalk:container:tomcat:jvmoptions` – このページで前述したこの名前空間のオプションに加えて、以前の Amazon Linux AMI プラットフォームバージョンでは以下もサポートされています。
  - `XX:MaxPermSize` – JVM のパーマネント領域の最大サイズ
- `aws:elasticbeanstalk:environment:proxy` – プロキシサーバーの選択に加えて、レスポンスの圧縮も設定します。

次の例の設定ファイルは、プロキシ名前空間の設定オプションの使用を示しています。

Example .ebextensions/tomcat-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    GzipCompression: 'true'
    ProxyServer: nginx
```

Elastic Beanstalk 設定ファイルを含める — Amazon Linux AMI (AL1)

`.ebextensions` 設定ファイルをデプロイするには、アプリケーションソースに含めます。単一のアプリケーションについては、次のコマンドを実行して、圧縮された WAR ファイルに `.ebextensions` を追加します。

Example

```
zip -ur your_application.war .ebextensions
```

複数の WAR ファイルを必要とするアプリケーションの場合は、詳細な手順について、「Tomcat 環境用に複数の WAR ファイルをバンドルする」を参照してください。

Tomcat 環境用に複数の WAR ファイルをバンドルする

ウェブアプリが複数のウェブアプリケーションコンポーネントで構成されている場合は、コンポーネントごとに別の環境を実行する代わりに 1 つの環境でコンポーネントを実行してデプロイを簡素化し、運用コストを減らすことができます。この戦略は、多くのリソースを必要としない軽量のアプリケーションや、開発環境およびテスト環境で有効です。

環境に複数のウェブアプリケーションをデプロイするには、各コンポーネントのウェブアプリケーションアーカイブ (WAR) ファイルを 1 つのソースバンドルに組み合わせます。

複数の WAR ファイルを含むアプリケーションソースバンドルを作成するには、次の構造を使用して WAR ファイルを整理します。

```
MyApplication.zip
### .ebextensions
### .platform
### foo.war
### bar.war
### ROOT.war
```

複数の WAR ファイルを含むソースバンドルを AWS Elastic Beanstalk 環境にデプロイすると、ルートドメイン名とは異なるパスから各アプリケーションにアクセスできます。前述の例には、`foo`、`bar`、`ROOT` の 3 つのアプリケーションが含まれています。`ROOT.war` は、ルートドメインでアプリケーションを実行するように Elastic Beanstalk に指示する特殊なファイル名です。したがって、これら 3 つのアプリケーションには `http://MyApplication.elasticbeanstalk.com/foo`、`http://MyApplication.elasticbeanstalk.com/bar`、`http://MyApplication.elasticbeanstalk.com` でアクセスできます。

ソースバンドルには、WAR ファイル、オプションの `.ebextensions` フォルダ、およびオプションの `.platform` フォルダを含めることができます。これらのオプションの設定フォルダの詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

環境を起動するには (コンソール)

1. この事前に設定されたリンク: console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced を使用して、Elastic Beanstalk コンソールを開きます。
2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチか、コンテナベースアプリケーション用の Docker プラットフォームを選択します。
3. [アプリケーションコード] で、[コードのアップロード] を選択します。
4. ローカルファイル を選択し、[Choose file (ファイルの選択)] を選択して、ソースバンドルを開きます。
5. 確認と起動 を選択します。

6. 使用できる設定を確認し、アプリの作成 を選択します。

ソースバンドルの作成の詳細については、「Elastic Beanstalk アプリケーションソースバンドルを作成する」を参照してください。

プロジェクトフォルダーを構築する

Tomcat サーバーにデプロイされた場合に機能するように、コンパイルされた Java プラットフォーム Enterprise Edition (Java EE) ウェブアプリケーションアーカイブ (WAR ファイル) は、特定のガイドラインに従って構築される必要があります。プロジェクトのディレクトリが同じ基準を満たす必要はありませんが、同じ方法でプロジェクトのディレクトリを構築し、コンパイルやパッケージングを簡素化することが推奨されます。WAR ファイルコンテンツなどのプロジェクトフォルダを構築すると、ファイルがどのように関連付けられていて、ウェブサーバーでどのように動作するかを理解するのにも役立ちます。

次の推奨階層では、ウェブアプリケーションのソースコードは、[src] ディレクトリに配置され、構築スクリプトや構築スクリプトが生成する WAR ファイルから隔離されます。

```
~/workspace/my-app/
|-- build.sh            - Build script that compiles classes and creates a WAR
|-- README.MD           - Readme file with information about your project, notes
|-- ROOT.war            - Source bundle artifact created by build.sh
`-- src                 - Source code folder
    |-- WEB-INF         - Folder for private supporting files
    |   |-- classes     - Compiled classes
    |   |-- lib         - JAR libraries
    |   |-- tags        - Tag files
    |   |-- tlds        - Tag Library Descriptor files
    |   `-- web.xml     - Deployment Descriptor
    |-- com             - Uncompiled classes
    |-- css             - Style sheets
    |-- images          - Image files
    |-- js              - JavaScript files
    `-- default.jsp     - JSP (JavaServer Pages) webpage
```

[src] フォルダの内容は、サーバーにパッケージングしてデプロイする内容と一致します ([com] フォルダは例外です) 。com フォルダには、コンパイルされていないクラス (.java ファイル) が含まれます。これらをコンパイルし、アプリケーションコードからアクセスできるように WEB-INF/classes ディレクトリに配置する必要があります。

WEB-INF ディレクトリには、ウェブサーバー上でパブリックに動作しないコードや設定が含まれます。ソースディレクトリのルートの他のフォルダ (css、images、js) はウェブサーバー上の対応するパスで一般公開されます。

次の例は、前述のプロジェクトディレクトリと同一ですが、ファイルとサブディレクトリの数が多い点のみが異なります。このプロジェクト例には、シンプルなタグ、モデル、サポートクラス、さらに record リソースのための Java サーバーページ (JSP) が含まれます。さらに、Bootstrap 用のスタイルシートと JavaScript、デフォルトの JSP ファイル、および 404 エラーのエラーページが含まれます。

[WEB-INF/lib] には、PostgreSQL の Java Database Connectivity(JDBC) ドライバを含む Java アーカイブ (JAR) ファイルが含まれます。クラスファイルがまだコンパイルされていないため、[WEB-INF/classes] は空です。

```
~/workspace/my-app/
|-- build.sh
|-- README.MD
|-- ROOT.war
`-- src
    |-- WEB-INF
    |   |-- classes
    |   |-- lib
    |   |   `-- postgresql-9.4-1201.jdbc4.jar
    |   |-- tags
    |   |   `-- header.tag
    |   |-- tlds
    |   |   `-- records.tld
    |   `-- web.xml
    |-- com
    |   `-- myapp
    |       |-- model
    |       |   `-- Record.java
    |       `-- web
    |           `-- ListRecords.java
    |-- css
    |   |-- bootstrap.min.css
    |   `-- myapp.css
    |-- images
    |   `-- myapp.png
    |-- js
    |   `-- bootstrap.min.js
    |-- 404.jsp
```

```
    |-- default.jsp
    `-- records.jsp
```

シェルスクリプトを使用して WAR ファイルを構築する

[build.sh] は、Java クラスをコンパイルする非常にシンプルなシェルスクリプトです。また、WAR ファイルを構築してローカルテストのために Tomcat の [webapps] ディレクトリにコピーします。

```
cd src
javac -d WEB-INF/classes com/myapp/model/Record.java
javac -classpath WEB-INF/lib/*:WEB-INF/classes -d WEB-INF/classes com/myapp/model/
Record.java
javac -classpath WEB-INF/lib/*:WEB-INF/classes -d WEB-INF/classes com/myapp/web/
ListRecords.java

jar -cvf ROOT.war *.jsp images css js WEB-INF
cp ROOT.war /Library/Tomcat/webapps
mv ROOT.war ../
```

WAR ファイルでは、前述の例の [src] ディレクトリ ([src/com] フォルダを除きます) に存在する同じストラクチャを探します。jar コマンドは自動的に META-INF/MANIFEST.MF ファイルを作成します。

```
~/workspace/my-app/ROOT.war
|-- META-INF
|   `-- MANIFEST.MF
|-- WEB-INF
|   |-- classes
|   |   `-- com
|   |       `-- myapp
|   |           |-- model
|   |           |   `-- Records.class
|   |           `-- web
|   |               `-- ListRecords.class
|   |-- lib
|   |   `-- postgresql-9.4-1201.jdbc4.jar
|   |-- tags
|   |   `-- header.tag
|   |-- tlds
|   |   `-- records.tld
|   `-- web.xml
```

```
|-- css
|   |-- bootstrap.min.css
|   `-- myapp.css
|-- images
|   `-- myapp.png
|-- js
|   `-- bootstrap.min.js
|-- 404.jsp
|-- default.jsp
`-- records.jsp
```

`.gitignore` を使用する

コンパイルされたクラスファイルと WAR ファイルが Git レポジトリにコミットされる、または Git コマンドの実行時にこれらのファイルに関するメッセージが表示されることを防ぐため、プロジェクトフォルダの `.gitignore` という名前のファイルに関連ファイルタイプを追加します。

~/workspace/myapp/.gitignore

```
*.zip
*.class
```

プロキシサーバーを設定します

Tomcat プラットフォームはリバースプロキシとして nginx (デフォルト) または Apache HTTP Server を使用し、インスタンスのポート 80 から、ポート 8080 でリッスンしている Tomcat ウェブコンテナにリクエストを中継します。Elastic Beanstalk では、デフォルトのプロキシ設定が用意されています。これは拡張することも、独自の設定で完全に上書きすることもできます。

ご使用のプラットフォームバージョンでプロキシサーバーを設定する

すべての AL2023/AL2 プラットフォームでは、統一されたプロキシ設定機能がサポートされています。AL2023/AL2 を実行中のプラットフォームバージョンでプロキシサーバーを設定する方法の詳細については、「リバースプロキシの設定」を参照してください。

(Amazon Linux 2 より前の) Amazon Linux AMI Tomcat プラットフォームでのプロキシの設定

Elastic Beanstalk Tomcat 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションの追加情報をお読みください。

メモ

- このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
- 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

Tomcat 環境のプロキシサーバーの選択 — Amazon Linux AMI (AL1)

(Amazon Linux 2 より前の) Amazon Linux AMI に基づく Tomcat プラットフォームバージョンは、デフォルトでプロキシに Apache 2.4 を使用します。ソースコードに設定ファイルを含めることにより、Apache 2.2 または nginx を使用する選択ができます。次の例では、nginx が使用されるように Elastic Beanstalk を設定しています。

Example .ebextensions/nginx-proxy.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: nginx
```

Apache 2.2 から Apache 2.4 への移行 — Amazon Linux AMI (AL1)

Apache 2.2 向けに開発されたアプリケーションがある場合、Apache 2.4 への移行について理解するには、このセクションをお読みください。

Java with Tomcat プラットフォームの更新 (2018 年 5 月 24 日) とともにリリースされた Tomcat プラットフォームバージョン 3.0.0 の以降の設定では、Tomcat プラットフォームのデフォルトのプロキシは Apache 2.4 です。Apache 2.4 の `.conf` ファイルは、Apache 2.2 のものと完全な下位互換性がありません。Elastic Beanstalk には、各 Apache バージョンで正しく動作するデフォルトの `.conf` ファイルが 1 つ含まれています。「デフォルトの Apache 設定の拡張および上書き — Amazon Linux AMI (AL1)」で説明されているように、アプリケーションが Apache の設定をカスタマイズしない場合は、Apache 2.4 への移行は問題ありません。

アプリケーションが Apache の設定を拡張または上書きする場合は、Apache 2.4 に移行するためにいくつかの変更を加える必要があります。詳細は、Apache Software Foundation サイトの 2.2 から 2.4 へのアップグレードを参照してください。Apache 2.4 への移行が正常に完了するまで、一時的な対策として、次の設定ファイルをソースコードに含めることで、アプリケーションで Apache 2.2 を使用することができます。

Example .ebextensions/apache-legacy-proxy.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache/2.2
```

簡単に修正するには、Elastic Beanstalk コンソールでプロキシサーバーを選択することもできます。

Elastic Beanstalk コンソールで Tomcat 環境にプロキシを選択するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [プロキシサーバー] を選択し、[`Apache 2.2 (deprecated)`] を選択します。
6. ページの最下部で [適用] を選択し変更を保存します。

デフォルトの Apache 設定の拡張および上書き — Amazon Linux AMI (AL1)

追加の設定ファイルを使用して、Elastic Beanstalk のデフォルト Apache 設定を拡張できます。または、Elastic Beanstalk のデフォルトの Apache 設定を完全に上書きすることもできます。

> **Note**
>
> - すべての Amazon Linux 2 プラットフォームでは、統一されたプロキシ設定の特徴が support されています。Amazon Linux 2 を実行中の Tomcat プラットフォームバージョンでプロキシサーバーを設定する方法の詳細については、「リバースプロキシの設定」を参照してください。

- Elastic Beanstalk アプリケーションを Amazon Linux 2 プラットフォームに移行する場合は、「the section called “AL2023/AL2 への移行”」の情報も必ずお読みください。

Elastic Beanstalk のデフォルトの Apache 設定を拡張するには、アプリケーションソースバンドルの `.conf` というフォルダに `.ebextensions/httpd/conf.d` 設定ファイルを追加します。Elastic Beanstalk の Apache 設定では、このフォルダに `.conf` ファイルが自動的に含められます。

```
~/workspace/my-app/
|-- .ebextensions
|   -- httpd
|      -- conf.d
|         -- myconf.conf
|         -- ssl.conf
-- index.jsp
```

たとえば、次の Apache 2.4 設定では、ポート 5000 にリスナーを追加します。

Example .ebextensions/httpd/conf.d/port5000.conf

```
listen 5000
<VirtualHost *:5000>
  <Proxy *>
    Require all granted
  </Proxy>
  ProxyPass / http://localhost:8080/ retry=0
  ProxyPassReverse / http://localhost:8080/
  ProxyPreserveHost on

  ErrorLog /var/log/httpd/elasticbeanstalk-error_log
</VirtualHost>
```

Elastic Beanstalk のデフォルトの Apache 設定を完全に上書きするには、ソースバンドルの `.ebextensions/httpd/conf/httpd.conf` に設定を含めます。

```
~/workspace/my-app/
|-- .ebextensions
|   `-- httpd
|       `-- conf
|           `-- httpd.conf
`-- index.jsp
```

Elastic Beanstalk の Apache 設定を上書きするには、`httpd.conf` に以下の行を追加することにより、Elastic Beanstalk でのヘルスレポートとモニタリングの強化、レスポンスの圧縮、および静的ファイルに関して Elastic Beanstalk の設定を適用します。

```
IncludeOptional conf.d/*.conf
IncludeOptional conf.d/elasticbeanstalk/*.conf
```

お使いの環境で Apache 2.2 をプロキシとして使用している場合は、`IncludeOptional` ディレクティブを `Include` に置き換えます。2 つの Apache バージョンにおけるこれら 2 つのディレクティブの動作の詳細については、Include in Apache 2.4、IncludeOptional in Apache 2.4、および Include in Apache 2.2 を参照してください。

> **Note**
>
> ポート 80 のデフォルトのリスナーを上書きするには、`00_application.conf` というファイルを `.ebextensions/httpd/conf.d/elasticbeanstalk/` に含めて Elastic Beanstalk の設定を上書きします。

実例については、環境内のインスタンスの `/etc/httpd/conf/httpd.conf` にある Elastic Beanstalk のデフォルト設定ファイルを参照してください。ソースバンドルの `.ebextensions/httpd` フォルダのすべてのファイルは、デプロイ中に `/etc/httpd` にコピーされます。

デフォルトの nginx 設定の拡張および上書き — Amazon Linux AMI (AL1)

Elastic Beanstalk のデフォルトの nginx 設定を拡張するには、アプリケーションソースバンドル内の `.conf` というフォルダに `.ebextensions/nginx/conf.d/` 設定ファイルを追加します。Elastic Beanstalk の nginx 設定では、このフォルダに `.conf` ファイルが自動的に含められます。

```
~/workspace/my-app/
|-- .ebextensions
|   `-- nginx
|       `-- conf.d
|           |-- elasticbeanstalk
|           |   `-- my-server-conf.conf
|           `-- my-http-conf.conf
`-- index.jsp
```

conf.d フォルダの拡張子が .conf であるファイルが、デフォルト設定の http ブロックに含まれます。conf.d/elasticbeanstalk フォルダのファイルは、server ブロック内の http ブロックに含まれます。

Elastic Beanstalk のデフォルトの nginx 設定を完全に上書きするには、ソースバンドルの .ebextensions/nginx/nginx.conf に設定を含めます。

```
~/workspace/my-app/
|-- .ebextensions
|   `-- nginx
|       `-- nginx.conf
`-- index.jsp
```

メモ

- Elastic Beanstalk の nginx 設定を上書きするには、設定の server ブロックに以下の行を追加することにより、ポート 80 のリスナー、レスポンスの圧縮、および静的ファイルに関して Elastic Beanstalk の設定を適用します。

  ```
  include conf.d/elasticbeanstalk/*.conf;
  ```

- ポート 80 のデフォルトのリスナーを上書きするには、00_application.conf というファイルを .ebextensions/nginx/conf.d/elasticbeanstalk/ に含めて Elastic Beanstalk の設定を上書きします。
- また、設定の http ブロックに以下の行を含めることにより、Elastic Beanstalk でのヘルスレポートとモニタリングの強化 およびログ記録に関して Elastic Beanstalk の設定を適用します。

  ```
      include       conf.d/*.conf;
  ```

実例については、環境内のインスタンスの /etc/nginx/nginx.conf にある Elastic Beanstalk のデフォルト設定ファイルを参照してください。ソースバンドルの .ebextensions/nginx フォルダのすべてのファイルは、デプロイ中に /etc/nginx にコピーされます。

Elastic Beanstalk Java SE プラットフォームの使用

このトピックでは、Java SE プラットフォームで実行される AWS Elastic Beanstalk Java アプリケーションを設定、構築、実行する方法について説明します。

Elastic Beanstalk Java SE プラットフォームは、コンパイルされた JAR ファイルから独自に Java ウェブアプリケーションを実行するためのプラットフォームバージョンのセットです。インスタンス内でコンパイルするには、アプリケーションをローカルでコンパイルするか、ビルドスクリプトを使用してソースコードをアップロードします。Java SE プラットフォームバージョンはプラットフォームブランチにグループ化され、各ブランチは Java の 1 つのメジャーバージョンに対応します。

> **Note**
>
> Elastic Beanstalk では、アプリケーションの JAR ファイルの解析は行われません。Elastic Beanstalk に必要なファイルは、JAR ファイルの外部に保存します。たとえば、ワーカー環境の `cron.yaml` ファイルはアプリケーションのソースバンドルのルート (JAR ファイルの横) に含めます。

設定オプションは実行中の環境の設定を変更するために Elastic Beanstalk コンソールで利用できます。環境を終了したときにその設定が失われないようにするため、保存済み設定を使用して設定を保存し、それを後で他の環境に適用することができます。

ソースコードの設定を保存する場合、設定ファイルを含めることができます。設定ファイルの設定は、環境を作成するたびに、またはアプリケーションをデプロイするたびに適用されます。設定ファイルを使用して、デプロイの間にパッケージをインストールしたり、スクリプトを実行したり、他のインスタンスのカスタマイズオペレーションを実行することもできます。

Elastic Beanstalk Java SE プラットフォームには、リバースプロキシとして機能する nginx サーバーが含まれています。このサーバーは、キャッシュ型静的コンテンツを提供し、アプリケーションにリクエストを渡します。このプラットフォームには、アプリケーションの負荷を減らすため、ソースコードのフォルダから静的アセットに対応するようプロキシサーバーを設定する設定オプションが用意されています。高度なシナリオでは、ソースバンドルに独自の .conf ファイルを含めて Elastic Beanstalk のプロキシ設定を拡張するか、これを完全に上書きできます。

アプリケーションソースに (ソースバンドル内ではなく、単独で) 単一の JAR ファイルだけを提供した場合は、JAR ファイルの名前が Elastic Beanstalk によって `application.jar` に変更され、`java -jar application.jar` を使用して実行されます。環境でサーバーインスタンスを実

行するプロセスを設定するには、オプションの Procfile をソースバンドルに含めます。ソースバンドルのルートに JAR が 2 つ以上ある場合、または java コマンドをカスタマイズして JVM オプションを設定する場合は `Procfile` が必要です。

ソースバンドルには、アプリケーションとともに常に `Procfile` を指定することをお勧めします。このようにして、アプリケーションに対して Elastic Beanstalk が実行するプロセスと、これらのプロセスが受け取る引数を正確に制御できます。

Java クラスをコンパイルし、デプロイ時に環境内の EC2 インスタンスでの他のビルドコマンドを実行するには、アプリケーションバンドルに Buildfile を含めます。`Buildfile` により、JAR をローカルにコンパイルする代わりに、ソースコードをそのままデプロイしてサーバー上に構築できます。Java SE プラットフォームには、サーバー上でのビルドを可能にする共通ビルドツールが含まれます。

Elastic Beanstalk Linux ベースのプラットフォームを拡張するさまざまな方法の詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

Java SE 環境を設定する

Java SE プラットフォーム設定では、Amazon EC2 インスタンスの動作を微調整できます。Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を編集できます。

Elastic Beanstalk コンソールを使用して、Amazon S3 へのログローテーションを有効にでき、アプリケーションが環境から読むことができる変数を設定します。

Elastic Beanstalk コンソールで Java SE 環境を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

ログオプション

[Log Options (ログオプション)] セクションには、2 つの設定があります。

- インスタンスプロファイル – アプリケーションに関連付けられた Amazon S3 バケットへのアクセス許可が付与されているインスタンスプロファイルを指定します。

- [Enable log file rotation to Amazon S3] (Amazon S3 へのログファイルのローテーションの有効化) - アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーするかどうかを指定します。

静的ファイル

パフォーマンスを向上させるために、[Static files] (静的ファイル) セクションを使用して、ウェブアプリケーション内のディレクトリセットから静的ファイル (HTML、イメージなど) を配信するようにプロキシサーバーを設定することができます。ディレクトリごとに、仮想パスをディレクトリマッピングに設定します。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接　　処理します。

設定ファイルまたは、Elastic Beanstalk コンソールを使用した静的ファイルの設定の詳細については、「the section called “静的ファイル”」を参照してください。

環境プロパティ

環境プロパティ セクションでは、アプリケーションを実行している Amazon EC2 インスタンスの環境設定を指定できます。環境プロパティは、キーと値のペアでアプリケーションに渡されます。

Elastic Beanstalk で実行される Java SE 環境内では、`System.getenv()` 関数を使用して環境変数にアクセスできます。たとえば、次のコードを使用して変数に `API_ENDPOINT` という名前のプロパティを読み取ることができます。

```
String endpoint = System.getenv("API_ENDPOINT");
```

詳細については、「環境変数およびその他のソフトウェア設定」を参照してください。

Java SE 設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクを実行できます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

Java SE プラットフォームは、プラットフォーム固有の名前空間を定義しません。`aws:elasticbeanstalk:environment:proxy:staticfiles` 名前空間を使用して、静的ファイルを配信するようにプロキシを設定できます。詳細と例については、「the section called “静的ファイル”」を参照してください。

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については、「設定オプション」を参照してください。

(Amazon Linux 2 より前の) Amazon Linux AMI Java SE プラットフォーム

Elastic Beanstalk Java SE 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションの追加情報をお読みください。

> **メモ**
>
> - このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
> - 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

Java SE ネームスペース — Amazon Linux AMI (AL1)

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクをパフォーマンスできます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

Java SE プラットフォームは、すべてのプラットフォームでサポートされる名前空間の他に、プラットフォーム固有の設定の名前空間を 1 つサポートします。名前空間 `aws:elasticbeanstalk:container:java:staticfiles` により、ウェブアプリケーションのパスを、静的コンテンツを含むアプリケーションソースバンドルのフォルダにマッピングするオプションを定義できます。

たとえば、この option_settings スニペットは、静的ファイルの名前空間で 2 つのオプションを定義します。1 つめのオプションはパス `/public` を [public] というフォルダに、2 つめのオプションはパス `/images` を [img] というフォルダにマッピングします。

```
option_settings:
```

```
  aws:elasticbeanstalk:container:java:staticfiles:
    /html: statichtml
    /images: staticimages
```

この名前空間を使用してマッピングされるフォルダは、ソースバンドルのルートに実際に存在するフォルダであることが必要です。パスを JAR ファイルのフォルダにマッピングすることはできません。

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

Buildfile で JARs on-server を構築する

ソースバンドルの [Buildfile] ファイルからビルドコマンドを起動することで、環境内の EC2 インスタンスにアプリケーションのクラスファイルと JAR を構築できます。

`Buildfile` のコマンドは 1 回のみ実行され、完了後に終了される必要があります。一方、Procfile のコマンドは、アプリケーションが有効な間は継続的に実行される必要があり、終了された場合には再起動されます。アプリケーションで JAR を実行するには、`Procfile` を使用します。

`Buildfile` の配置と構文の詳細については、「ビルドファイルと Procfile」を参照してください。

以下の `Buildfile` の例は、Apache Maven を実行してソースコードからウェブアプリケーションを構築しています。この機能を使用するサンプルアプリケーションについては、Java ウェブアプリケーション例を参照してください。

Example [Buildfile]

```
build: mvn assembly:assembly -DdescriptorId=jar-with-dependencies
```

Java SE プラットフォームには、ビルドスクリプトから起動できる次のビルドツールが含まれます。

- `javac` – Java コンパイラー
- `ant` – Apache Ant
- `mvn` – Apache Maven
- `gradle` – Gradle

[Procfile] でアプリケーションプロセスを設定します

アプリケーションソースバンドルのルートに JAR ファイルが複数ある場合は、どの JAR を実行するかを Elastic Beanstalk に伝える `Procfile` ファイルを含める必要があります。単一の JAR アプリケーション用の [`Procfile`] ファイルを含め、アプリケーションを実行する Java 仮想マシン（JVM）を設定することもできます。

ソースバンドルには、アプリケーションとともに常に `Procfile` を指定することをお勧めします。このようにして、アプリケーションに対して Elastic Beanstalk が実行するプロセスと、これらのプロセスが受け取る引数を正確に制御できます。

`Procfile` の書き込みと使用の詳細については、「ビルドファイルと Procfile」を参照してください。

Example [Procfile]

```
web: java -Xms256m -jar server.jar
cache: java -jar mycache.jar
web_foo: java -jar other.jar
```

アプリケーション内の主要な JAR を実行するコマンドは、[`web`] と呼ばれ、`Procfile` 内のコマンドリストの最初に記載されている必要があります。nginx サーバーは、環境のロードバランサーから受信するすべての HTTP リクエストをアプリケーションに転送します。

Elastic Beanstalk は、Procfile 内のすべてのエントリが常に実行されている必要があるとみなし、Procfile に定義されたアプリケーションが終了した場合には自動的に再起動します。終了後に再起動の必要がないコマンドを実行するには、`Buildfile` を使用します。

(Amazon Linux 2 より前の) Amazon Linux AMI での Procfile の使用

Elastic Beanstalk Java SE 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションの追加情報をお読みください。

> **メモ**
>
> - このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
> - 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポート

されている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

ポートパッシング — Amazon Linux AMI (AL1)

デフォルトでは、Elastic Beanstalk はポート 5000 でアプリケーションにリクエストを送信するように nginx プロキシを設定します。デフォルトのポートを上書きするには、`PORT` 環境プロパティを、主要なアプリケーションがリッスンするポートに設定します。

`Procfile` を使用して複数のアプリケーションを実行する場合、Amazon Linux AMI プラットフォームバージョン上の Elastic Beanstalk では、各アプリケーションがリッスンするポートとして、1 つ前のアプリケーションのポート番号に 100 を加算した番号のポートが想定されます。各アプリケーション内からアクセス可能な PORT 変数には、そのアプリケーションの実行が予想されるポート番号が Elastic Beanstalk によって設定されます。`System.getenv("PORT")` を呼び出すことで、アプリケーションコード内のこの変数にアクセスできます。

以前の `Procfile` 例では、`web` アプリケーションはポート 5000 を、`cache` はポート 5100 を、`web_foo` はポート 5200 をリッスンします。[web] は、`PORT` 変数を読み取ることでリスニングポートを設定し、ポート番号に 100 を足して [cache] がリッスンしているポートを決定し、リクエストを送信します。

プロキシサーバーを設定します

Elastic Beanstalk は、リバースプロキシとして nginx を使用し、ポート 80 の Elastic Load Balancing ロードバランサーにアプリケーションをマッピングします。Elastic Beanstalk では、デフォルトの nginx 設定が用意されています。これは拡張することも、独自の設定で完全に上書きすることもできます。

デフォルトでは、Elastic Beanstalk はポート 5000 でアプリケーションにリクエストを送信するように nginx プロキシを設定します。デフォルトのポートを上書きするには、`PORT` 環境プロパティを、主要なアプリケーションがリッスンするポートに設定します。

> **Note**
>
> アプリケーションがリッスンしているポートは、ロードバランサーからリクエストを受信するために nginx サーバーがリッスンするポートに影響を与えません。

ご使用のプラットフォームバージョンでプロキシサーバーを設定する

すべての AL2023/AL2 プラットフォームでは、統一されたプロキシ設定機能がサポートされています。AL2023/AL2 を実行中のプラットフォームバージョンでプロキシサーバーを設定する方法の詳細については、「リバースプロキシの設定」を参照してください。

(Amazon Linux 2 より前の) Amazon Linux AMI でのプロキシの設定

Elastic Beanstalk Java SE 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションの追加情報をお読みください。

> **メモ**
>
> - このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
> - 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

デフォルトのプロキシ設定の拡張および上書き — Amazon Linux AMI (AL1)

Elastic Beanstalk のデフォルトの nginx 設定を拡張するには、アプリケーションソースバンドル内の `.conf` というフォルダに `.ebextensions/nginx/conf.d/` 設定ファイルを追加します。Elastic Beanstalk の nginx 設定では、このフォルダに `.conf` ファイルが自動的に含められます。

```
~/workspace/my-app/
|-- .ebextensions
|   `-- nginx
|       `-- conf.d
|           `-- myconf.conf
`-- web.jar
```

Elastic Beanstalk のデフォルトの nginx 設定を完全に上書きするには、ソースバンドルの `.ebextensions/nginx/nginx.conf` に設定を含めます。

```
~/workspace/my-app/
|-- .ebextensions
|   `-- nginx
|       `-- nginx.conf
`-- web.jar
```

Elastic Beanstalk の nginx 設定を上書きするには、`nginx.conf` に以下の行を追加することにより、Elastic Beanstalk でのヘルスレポートとモニタリングの強化、自動アプリケーションマッピング、および静的ファイルに関して Elastic Beanstalk の設定を適用します。

```
include conf.d/elasticbeanstalk/*.conf;
```

以下に示す Scorekeep サンプルアプリケーションの設定例では、Elastic Beanstalk のデフォルトの設定をオーバーライドし、`public` の `/var/app/current`サブディレクトリにある静的ウェブアプリケーションを処理します。このサブディレクトリは、Java SE プラットフォームによってアプリケーションのソースコードがコピーされる場所です。`/api` の場所は、`/api/` のルートへのトラフィックを、ポート 5000 でリッスンしている Spring アプリケーションに転送します。他のすべてのトラフィックは、ルートパスでウェブアプリによって処理されます。

Example

```
user                    nginx;
error_log               /var/log/nginx/error.log warn;
pid                     /var/run/nginx.pid;
worker_processes        auto;
worker_rlimit_nofile    33282;

events {
    worker_connections  1024;
}

http {
  include       /etc/nginx/mime.types;
  default_type  application/octet-stream;

  log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"';

  include       conf.d/*.conf;
```

```
  map $http_upgrade $connection_upgrade {
      default     "upgrade";
  }

  server {
      listen        80 default_server;
      root /var/app/current/public;

      location / {
      }git pull


      location /api {
          proxy_pass          http://127.0.0.1:5000;
          proxy_http_version  1.1;

          proxy_set_header    Connection          $connection_upgrade;
          proxy_set_header    Upgrade             $http_upgrade;
          proxy_set_header    Host                $host;
          proxy_set_header    X-Real-IP           $remote_addr;
          proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
      }

      access_log    /var/log/nginx/access.log main;

      client_header_timeout 60;
      client_body_timeout   60;
      keepalive_timeout     60;
      gzip                  off;
      gzip_comp_level       4;

      # Include the Elastic Beanstalk generated locations
      include conf.d/elasticbeanstalk/01_static.conf;
      include conf.d/elasticbeanstalk/healthd.conf;
  }
}
```

Amazon RDS DB インスタンスを Java Elastic Beanstalk 環境に追加する

このトピックでは、Elastic Beanstalk コンソールを使用して Amazon RDS を作成する手順について説明します。アプリケーションで収集または変更したデータを保存するには、Amazon Relational

Database Service (Amazon RDS) DB インスタンスを使用します。データベースは、Elastic Beanstalk でお客様の環境にアタッチして管理したり、外部で作成して管理したりできます。

初めて Amazon RDS を使用する場合は、Elastic Beanstalk コンソールを使用してテスト環境に DB インスタンスを追加し、そのインスタンスにアプリケーションから接続できることを確認します。

お客様の環境に DB インスタンスを追加するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. DB エンジンを選択して、ユーザー名とパスワードを入力します。
6. ページの最下部で [適用] を選択し変更を保存します。

DB インスタンスの追加には約 10 分かかります。環境の更新が完了すると、DB インスタンスのホスト名とその他の接続情報は以下の環境プロパティを通じてアプリケーションに使用できるようになります。

プロパティ名	説明	プロパティ値
`RDS_HOSTNAME`	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
`RDS_PORT`	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。
`RDS_DB_NAME`	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。

プロパティ名	説明	プロパティ値
RDS_USERNAME	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。
RDS_PASSWORD	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールでは参照できません。

内部 DB インスタンスの設定の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。Elastic Beanstalk で使用する外部データベースを設定する手順については、「Amazon RDS で Elastic Beanstalk を使用する」を参照してください。

データベースに接続するには、適切なドライバーの JAR ファイルをアプリケーションに追加して、コードにドライバークラスをロードし、Elastic Beanstalk で提供される環境プロパティを使用して接続オブジェクトを作成します。

セクション

- JDBC ドライバーのダウンロード
- データベースへの接続 (Java SE プラットフォーム)
- データベースへの接続 (Tomcat プラットフォーム)
- データベース接続のトラブルシューティング

JDBC ドライバーのダウンロード

選択した DB エンジン用の JDBC ドライバーの JAR ファイルが必要になります。JAR ファイルをソースコードに保存し、データベースへの接続を作成するクラスをコンパイルするときのクラスパスに含めます。

以下の場所で DB エンジン用の最新のドライバが見つかります。

- MySQL – MySQL Connector/J
- Oracle SE-1 – Oracle JDBC ドライバー
- Postgres – PostgreSQL JDBC ドライバー
- SQL Server – Microsoft JDBC ドライバー

JDBC ドライバーを使用するには、コード内で `Class.forName()` により接続を作成する前に、`DriverManager.getConnection()` を呼び出してそのドライバーをロードします。

JDBC では次の形式の接続文字列が使用されます:

```
jdbc:driver://hostname:port/dbName?user=userName&password=password
```

ホスト名、ポート、データベース名、ユーザー名、パスワードは、Elastic Beanstalk からアプリケーションに提供される環境変数から取得できます。ドライバ名は、データベースタイプとドライババージョンによって異なります。以下はドライバー名の一例です。

- `mysql` for MySQL
- `postgresql` for PostgreSQL
- `oracle:thin` (Oracle Thin)
- `oracle:oci` (Oracle OCI)
- `oracle:oci8` (Oracle OCI 8)
- `oracle:kprb` (Oracle KPRB)
- `sqlserver` (SQL Server)

データベースへの接続 (Java SE プラットフォーム)

Java SE 環境では、`System.getenv()` を使用して環境から接続変数を読み取ります。以下のコード例では、PostgreSQL データベースへの接続を作成するクラスを示しています。

```
private static Connection getRemoteConnection() {
    if (System.getenv("RDS_HOSTNAME") != null) {
      try {
      Class.forName("org.postgresql.Driver");
      String dbName = System.getenv("RDS_DB_NAME");
      String userName = System.getenv("RDS_USERNAME");
      String password = System.getenv("RDS_PASSWORD");
      String hostname = System.getenv("RDS_HOSTNAME");
      String port = System.getenv("RDS_PORT");
      String jdbcUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbName + "?
user=" + userName + "&password=" + password;
      logger.trace("Getting remote connection with connection string from environment
 variables.");
      Connection con = DriverManager.getConnection(jdbcUrl);
```

```
      logger.info("Remote connection successful.");
      return con;
    }
    catch (ClassNotFoundException e) { logger.warn(e.toString());}
    catch (SQLException e) { logger.warn(e.toString());}
    }
    return null;
  }
```

データベースへの接続 (Tomcat プラットフォーム)

Tomcat 環境では、環境プロパティは、`System.getProperty()` でアクセス可能なシステムプロパティとして用意されています。

以下のコード例では、PostgreSQL データベースへの接続を作成するクラスを示しています。

```
private static Connection getRemoteConnection() {
    if (System.getProperty("RDS_HOSTNAME") != null) {
      try {
      Class.forName("org.postgresql.Driver");
      String dbName = System.getProperty("RDS_DB_NAME");
      String userName = System.getProperty("RDS_USERNAME");
      String password = System.getProperty("RDS_PASSWORD");
      String hostname = System.getProperty("RDS_HOSTNAME");
      String port = System.getProperty("RDS_PORT");
      String jdbcUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbName + "?
user=" + userName + "&password=" + password;
      logger.trace("Getting remote connection with connection string from environment
 variables.");
      Connection con = DriverManager.getConnection(jdbcUrl);
      logger.info("Remote connection successful.");
      return con;
    }
    catch (ClassNotFoundException e) { logger.warn(e.toString());}
    catch (SQLException e) { logger.warn(e.toString());}
    }
    return null;
  }
```

接続の取得時や SQL ステートメントの実行時に問題が発生する場合は、JSP ファイルに以下のコードを追加してみることができます。このコードは、DB インスタンスに接続し、テーブルを作成して、そのテーブルに書き込みます。

```
<%@ page import="java.sql.*" %>
<%
  // Read RDS connection information from the environment
  String dbName = System.getProperty("RDS_DB_NAME");
  String userName = System.getProperty("RDS_USERNAME");
  String password = System.getProperty("RDS_PASSWORD");
  String hostname = System.getProperty("RDS_HOSTNAME");
  String port = System.getProperty("RDS_PORT");
  String jdbcUrl = "jdbc:mysql://" + hostname + ":" +
    port + "/" + dbName + "?user=" + userName + "&password=" + password;

  // Load the JDBC driver
  try {
    System.out.println("Loading driver...");
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("Driver loaded!");
  } catch (ClassNotFoundException e) {
    throw new RuntimeException("Cannot find the driver in the classpath!", e);
  }

  Connection conn = null;
  Statement setupStatement = null;
  Statement readStatement = null;
  ResultSet resultSet = null;
  String results = "";
  int numresults = 0;
  String statement = null;

  try {
    // Create connection to RDS DB instance
    conn = DriverManager.getConnection(jdbcUrl);

    // Create a table and write two rows
    setupStatement = conn.createStatement();
    String createTable = "CREATE TABLE Beanstalk (Resource char(50));";
    String insertRow1 = "INSERT INTO Beanstalk (Resource) VALUES ('EC2 Instance');";
    String insertRow2 = "INSERT INTO Beanstalk (Resource) VALUES ('RDS Instance');";

    setupStatement.addBatch(createTable);
    setupStatement.addBatch(insertRow1);
    setupStatement.addBatch(insertRow2);
    setupStatement.executeBatch();
    setupStatement.close();
```

```
  } catch (SQLException ex) {
    // Handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
  } finally {
    System.out.println("Closing the connection.");
    if (conn != null) try { conn.close(); } catch (SQLException ignore) {}
  }

  try {
    conn = DriverManager.getConnection(jdbcUrl);

    readStatement = conn.createStatement();
    resultSet = readStatement.executeQuery("SELECT Resource FROM Beanstalk;");

    resultSet.first();
    results = resultSet.getString("Resource");
    resultSet.next();
    results += ", " + resultSet.getString("Resource");

    resultSet.close();
    readStatement.close();
    conn.close();

  } catch (SQLException ex) {
    // Handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
  } finally {
       System.out.println("Closing the connection.");
      if (conn != null) try { conn.close(); } catch (SQLException ignore) {}
  }
%>
```

結果を表示するには、JSP ファイルの HTML 部分の本文に次のコードを挿入します。

```
<p>Established connection to RDS. Read first two rows: <%= results %></p>
```

データベース接続のトラブルシューティング

アプリケーションからデータベースへの接続で問題が発生する場合は、ウェブコンテナのログとデータベースを確認します。

ログを確認する

Elastic Beanstalk 環境のすべてのログは、Eclipse 内から表示できます。 AWS Explorer ビューが開いていない場合は、ツールバーのオレンジ色の AWS アイコンの横にある矢印を選択し、 AWS Explorer ビューを表示を選択します。[AWS Elastic Beanstalk] および環境名を展開し、サーバーのコンテキストメニューを開きます (右クリック)。[Open in WTP Server Editor] を選択します。

[Server] ビューの [Log] タブを選択し、環境の集計ログを表示します。最新のログを開くには、ページ右上の [Refresh] ボタンを選択します。

下にスクロールし、`/var/log/tomcat7/catalina.out` 内の Tomcat ログを探します。これまでの例で何回かウェブページを読み込んだ場合は、次のように表示されることがあります。

```
-------------------------------------
/var/log/tomcat7/catalina.out
-------------------------------------
INFO: Server startup in 9285 ms
Loading driver...
Driver loaded!
SQLException: Table 'Beanstalk' already exists
SQLState: 42S01
VendorError: 1050
Closing the connection.
Closing the connection.
```

ウェブアプリケーションが標準出力に送信するすべての情報は、ウェブコンテナのログに表示されます。前の例では、アプリケーションは、ページが読み込まれるたびにテーブルを作成しようとします。これにより、最初のページ以降のすべてのページのロード時に SQL 例外が捕捉されます。

上記は例としては問題ありません。ただし、実際のアプリケーションでは、データベース定義をスキーマオブジェクトに保持して、モデルクラス内からトランザクションを実行し、コントローラーサーブレットによってリクエストを調整します。

RDS DB インスタンスに接続する

MySQL クライアントアプリケーションを使用すると、Elastic Beanstalk 環境内の RDS DB インスタンスに直接接続できます。

最初に、RDS DB インスタンスからセキュリティグループにアクセスできるようにして、コンピュータからのトラフィックを許可します。

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. [エンドポイント] の横にある Amazon RDS コンソールリンクを選択します。
6. [RDS Dashboard] のインスタンスの詳細ページで、[Security and Network] の [Security Groups] から、rds- で始まるセキュリティグループを選択します。

> **Note**
>
> データベースには、[Security Groups] のラベルが付けられたエントリが複数ある場合があります。古い方のアカウントにデフォルトの Amazon Virtual Private Cloud (Amazon VPC) が設定されてない場合のみ、最初のエントリ (awseb で始まるもの) を使用します。

7. [Security group details] で、[Inbound] タブ、[Edit] の順に選択します。
8. MySQL (ポート 3306)に、CIDR 形式で指定した IP アドレスからのトラフィックを許可するルールを追加します。
9. [Save] を選択します。変更はすぐに反映されます。

環境の Elastic Beanstalk 詳細設定に戻り、エンドポイントを書き留めます。RDS DB インスタンスに接続するには、ドメイン名を使用します。

MySQL クライアントをインストールし、ポート 3306 でデータベースへの接続を開始します。Windows の場合は、MySQL のホームページから MySQL Workbench をインストールし、プロンプトに従います。

Linux の場合は、お客様のディストリビューションに対応したパッケージマネージャを使用して、MySQL クライアントをインストールします。次の例は、Ubuntu および Debian から派生したその他の OS で利用できます。

```
// Install MySQL client
$ sudo apt-get install mysql-client-5.5
...
// Connect to database
$ mysql -h aas839jo2vwhwb.cnubrrfwfka8.us-west-2.rds.amazonaws.com -u username -
ppassword ebdb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 117
Server version: 5.5.40-log Source distribution
...
```

接続後、SQL コマンドを実行し、データベースのステータス、テーブルや行が作成されたかどうかなどの情報を確認できます。

```
mysql> SELECT Resource from Beanstalk;
+--------------+
| Resource     |
+--------------+
| EC2 Instance |
| RDS Instance |
+--------------+
2 rows in set (0.01 sec)
```

Java ツールとリソース

Java アプリケーションを開発する場合は、他にも参照できる場所がいくつかあります。

リソース	説明
Java AWS 開発フォーラム	質問を投稿してフィードバックを得ることができます。
Java デベロッパーセンター	サンプルコード、ドキュメント、ツール、追加リソースを 1 か所で入手できる場所です。

Node.js アプリケーションを Elastic Beanstalk でデプロイする

この章では、Node.js ウェブアプリケーションを設定してデプロイする手順について説明します AWS Elastic Beanstalk。また、データベース統合や Express フレームワークの使用などの一般的なタスクのチュートリアルも提供します。Elastic Beanstalk を使用すると、Amazon Web Services を使用して簡単に Node.js ウェブアプリケーションのデプロイ、管理、スケーリングができます。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) または Elastic Beanstalk コンソールを使用すると、わずか数分でアプリケーションをデプロイできます。Elastic Beanstalk アプリケーションをデプロイした後も、EB CLI を使用してアプリケーションと環境を管理することも、Elastic Beanstalk コンソール AWS CLIまたは APIs を使用することもできます。

EB CLI を使用して Hello World Node.js ウェブアプリケーションを作成してデプロイするには、「Node.js の QuickStart」のステップバイステップの手順に従います。

トピック

- QuickStart: Elastic Beanstalk に Node.js アプリケーションをデプロイする
- Elastic Beanstalk 用の Node.js 開発環境の設定
- Elastic Beanstalk Node.js プラットフォームを使用する
- Node.js のその他の Elastic Beanstalk サンプルアプリケーションとチュートリアル
- Node.js Express アプリケーションの Elastic Beanstalk へのデプロイ
- クラスタリング付き Node.js Express アプリケーションの Elastic Beanstalk へのデプロイ
- DynamoDB を使用して Node.js アプリケーションを Elastic Beanstalk にデプロイする
- Amazon RDS DB インスタンスを Node.js Elastic Beanstalk 環境に追加する
- Node.js ツールとリソース

QuickStart: Elastic Beanstalk に Node.js アプリケーションをデプロイする

この QuickStart チュートリアルでは、Node.js アプリケーションを作成し、 AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

> **Note**
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: Node.js アプリケーションを作成する
- ステップ 2: アプリケーションをローカルに実行する
- ステップ 3: EB CLI を使用して Node.js アプリケーションをデプロイする
- ステップ 4: Elastic Beanstalk でアプリケーションを実行する
- ステップ 5：クリーンアップ
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

すでに AWS アカウントをお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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、通話呼び出し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するように求められます。

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 Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 「 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

EB CLI

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

Node.js

Node.js ウェブサイトの「Node.js をインストールする方法」に従って、ローカルマシンに Node.js をインストールします。

次のコマンドを実行して、Node.js のインストールを確認します。

```
~$ node -v
```

ステップ 1: Node.js アプリケーションを作成する

プロジェクトディレクトリを作成します。

```
~$ mkdir eb-nodejs
~$ cd eb-nodejs
```

次に、Elastic Beanstalk を使用してデプロイするアプリケーションを作成します。ここでは、"Hello World" という RESTful ウェブサービスを作成します。

Example **~/eb-nodejs/server.js**

```
const http = require('node:http');

const hostname = '127.0.0.1';
const port = 8080;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello Elastic Beanstalk!\n');
});
```

```
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

このアプリケーションは、ポート 8080 でリスナーを開きます。Elastic Beanstalk は、デフォルトで Node.js のポート 8080 でアプリケーションにリクエストを転送します。

ステップ 2: アプリケーションをローカルに実行する

アプリケーションをローカルで実行するには、次のコマンドを実行します。

```
~/eb-nodejs$ node server.js
```

次のテキストが表示されます。

```
Server running at http://127.0.0.1:8080/
```

ウェブブラウザに URL アドレス `http://127.0.0.1:8080/` を入力します。ブラウザに「Hello Elastic Beanstalk!」と表示されます。

ステップ 3: EB CLI を使用して Node.js アプリケーションをデプロイする

次のコマンドを実行して、このアプリケーションの Elastic Beanstalk 環境を作成します。

環境を作成し、Node.js アプリケーションをデプロイするには

1. eb init コマンドを使用して EB CLI リポジトリを初期化します。

   ```
   ~/eb-nodejs$ eb init -p node.js nodejs-tutorial --region us-east-2
   ```

   このコマンドは、`nodejs-tutorial` という名前のアプリケーションを作成し、ローカルリポジトリを設定して最新の Node.js プラットフォームバージョンで環境を作成します。
2. (オプション) eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH を使用して connect できるようにします。

   ```
   ~/eb-nodejs$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   ```

```
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従ってキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。

3. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。Elastic Beanstalk は、アプリケーションの zip ファイルを自動的にビルドし、環境内の EC2 インスタンスにデプロイします。アプリケーションがデプロイされると、Elastic Beanstalk はポート 8080 でアプリケーションを起動します。

```
~/eb-nodejs$ eb create nodejs-env
```

Elastic Beanstalk が環境を作成するのに約 5 分かかります。

ステップ 4: Elastic Beanstalk でアプリケーションを実行する

環境を作成するプロセスが完了したら、eb open でウェブサイトを開きます。

```
~/eb-nodejs$ eb open
```

お疲れ様でした。Elastic Beanstalk で Node.js アプリケーションをデプロイしました。これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ Window が開きます。

ステップ 5：クリーンアップ

アプリケーションでの作業が終了したら、環境を終了できます。Elastic Beanstalk は、環境に関連付けられているすべての AWS リソースを終了します。

EB CLI を使用して Elastic Beanstalk 環境を終了するには、次のコマンドを実行します。

```
~/eb-nodejs$ eb terminate
```

AWS アプリケーションの リソース

1 つのインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありませ

ん。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えま

す。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

> **その他のチュートリアルを試す**
>
> 異なるアプリケーション例の他のチュートリアルを試したい場合は、「Node.js のその他の Elastic Beanstalk サンプルアプリケーションとチュートリアル」を参照してください。

1 つか 2 つのサンプルアプリケーションをデプロイし、ローカルで Node.js アプリケーションを開発して実行する準備が整ったら、「Elastic Beanstalk 用の Node.js 開発環境の設定」を参照します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。

Elastic Beanstalk 用の Node.js 開発環境の設定

このトピックでは、Node.js 開発環境を設定し、アプリケーションを AWS Elastic Beanstalkにデプロイする前にローカルでテストする手順について説明します。また、便利なツールのインストール手順を提供するウェブサイトも参照します。

トピック

- Node.js をインストールします
- npm のインストールを確認します
- AWS SDK for Node.js をインストールする
- Express ジェネレーターをインストールする
- Express フレームワークとサーバーを設定する

Node.js をインストールします

Node.js アプリケーションをローカルで実行するように Node.js をインストールします。指定しない場合は、Elastic Beanstalk がサポートする最新バージョンを取得します。サポートされているバージョンの一覧については、AWS Elastic Beanstalk プラットフォームドキュメントの「Node.js」を参照してください。

nodejs.org で Node.js をダウンロードします。

npm のインストールを確認します

Node.js では npm パッケージマネージャーを使用することで、アプリケーション用のツールやフレームワークのインストールが簡単に行えます。npm は Node.js と共に配信されるため、Node.js をダウンロードしてインストールする際に自動的にインストールされます。npm がインストールされていることを確認するには、次のコマンドを実行します。

```
$ npm -v
```

npm の詳細については、npmjs のウェブサイトを参照してください。

AWS SDK for Node.js をインストールする

アプリケーション内から AWS リソースを管理する必要がある場合は、Node.js に AWS SDK for JavaScript をインストールします。次のように npm を使用して SDK をインストールします。

```
$ npm install aws-sdk
```

詳細については、「AWS SDK for JavaScript in Node.js」のホームページにアクセスしてください。

Express ジェネレーターをインストールする

Express は、Node.js を実行するウェブアプリケーションフレームワークです。これを使用するには、まず Express ジェネレーターコマンドラインアプリケーションをインストールします。Express ジェネレーターがインストールされたら、express コマンドを実行してウェブアプリケーションのベースプロジェクト構造を生成できます。ベースプロジェクト、ファイル、および依存関係がインストールされたら、開発マシン上でローカル Express サーバーを起動できます。

> **Note**
>
> - 次のステップでは、Linux オペレーティングシステム上に Express ジェネレーターをインストールする手順を説明します。
> - Linux では、システムディレクトリに対する許可レベルに応じて、これらのコマンドの一部のプレフィックスに sudo が必要になる場合があります。

Express ジェネレーターを開発環境にインストールするには

1. Express フレームワークとサーバーの作業ディレクトリを作成します。

```
~$ mkdir node-express
~$ cd node-express
```

2. express コマンドにアクセスできるよう、Express をグローバルにインストールします。

```
~/node-express$ npm install -g express-generator
```

3. オペレーションシステムによっては、express コマンドを実行するパスを設定する必要があります。前のステップの出力は、パス変数を設定する必要がある場合の情報を提供します。Linux の例を次に示します。

```
~/node-express$ export PATH=$PATH:/usr/local/share/npm/bin/express
```

この章のチュートリアルに従うときは、別のディレクトリから express コマンドを実行する必要があります。各チュートリアルは、独自のディレクトリにベース Express プロジェクト構造を設定します。

これで、Express コマンドラインジェネレーターがインストールされました。これを使用して、ウェブアプリケーションのフレームワークディレクトリを作成し、依存関係を設定し、ウェブアプリケーションサーバーを起動できます。次に、作成した node-express ディレクトリでこれを実現するステップを実行します。

Express フレームワークとサーバーを設定する

次のステップに従って、ベース Express フレームワークのディレクトリとコンテンツを作成します。この章のチュートリアルには、チュートリアルの各アプリケーションディレクトリにベース Express フレームワークを設定するための次のステップも含まれています。

Express フレームワークとサーバーを設定するには

1. express コマンドを実行します。これによって、package.json と app.js、およびいくつかのディレクトリが生成されます。

```
~/node-express$ express
```

プロンプトが表示されたら、続行するには **y** と入力します。

2. ローカルの依存関係を設定します。

```
~/node-express$ npm install
```

3. ウェブアプリサーバーが起動することを確認します。

```
~/node-express$ npm start
```

次のような出力が表示されます:

```
> nodejs@0.0.0 start /home/local/user/node-express
> node ./bin/www
```

サーバーは、デフォルトでポート 3000 で実行されます。テストするには、別のターミナルで `curl http://localhost:3000` を実行するか、ローカルコンピュータでブラウザを開いて URL アドレス `http://localhost:3000` を入力します。

サーバーを停止するには、[Ctrl+C] を押します。

Elastic Beanstalk Node.js プラットフォームを使用する

このトピックでは、Elastic Beanstalk で Node.js アプリケーションを設定、ビルド、実行する方法について説明します。

AWS Elastic Beanstalk は、Node.js プログラミング言語のさまざまなバージョンの多数のプラットフォームブランチをサポートしています。完全なリストについては、「AWS Elastic Beanstalk プラットフォーム」ドキュメントの「Node.js」を参照してください。

Elastic Beanstalk には、Elastic Beanstalk 環境内の EC2 インスタンスで実行されるソフトウェアのカスタマイズに使用できる設定オプションが用意されています。アプリケーションに必要な環境変数を設定し、Amazon S3 に対してログのローテーションを有効にしたら、アプリケーションの出典で静的ファイルが含まれるフォルダを、プロキシサーバーによって提供されるパスにマッピングできます。

設定オプションは実行中の環境の設定を変更するために Elastic Beanstalk コンソールで利用できます。環境を終了したときにその設定が失われないようにするため、保存済み設定を使用して設定を保存し、それを後で他の環境に適用することができます。

ソースコードの設定を保存する場合、設定ファイルを含めることができます。設定ファイルの設定は、環境を作成するたびに、またはアプリケーションをデプロイするたびに適用されます。設定ファイルを使用して、デプロイの間にパッケージをインストールしたり、スクリプトを実行したり、他のインスタンスのカスタマイズオペレーションを実行することもできます。

ソースバンドルに `Package.json` ファイルを含めて、デプロイ中にパッケージをインストールしたり、開始コマンドを指定したり、アプリケーションで使用する Node.js のバージョンを指定したりできます。依存関係のバージョンをロックダウンする `npm-shrinkwrap.json` ファイルを含めることができます。

Node.js プラットフォームには、静的なアセットを配信し、アプリケーションにトラフィックを転送して、レスポンスを圧縮するためのプロキシサーバーが含まれています。アドバンスト・シナリオでは、デフォルトのプロキシ設定を拡張また上書きできます。

アプリケーションをスタートするには、いくつかのオプションがあります。ソースバンドルに Procfile を追加すると、アプリケーションをスタートするコマンドを指定できます。`Procfile` を指定しない場合、`package.json` ファイルを指定すると Elastic Beanstalk で `npm start` を実行します。そのいずれかを指定しない場合、Elastic Beanstalk では `app.js` または `server.js` をこの順序で探して、スクリプトを実行します。

Elastic Beanstalk コンソールで適用される設定は、設定ファイルに同じ設定があれば、それらの設定を上書きします。これにより、設定ファイルでデフォルト設定を定義し、コンソールでそのデフォルト設定を環境固有の設定で上書きできます。設定の優先順位の詳細と設定の他の変更方法については、「設定オプション」を参照してください。

Elastic Beanstalk Linux ベースのプラットフォームを拡張するさまざまな方法の詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

Node.js 環境の設定

Node.js プラットフォーム設定を使用して、Amazon EC2 インスタンスの動作を微調整できます。Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境での Amazon EC2 インスタンスの設定を編集できます。

Elastic Beanstalk コンソールを使用して、Amazon S3 へのログローテーションを有効にでき、アプリケーションが環境から読むことができる変数を設定します。

Elastic Beanstalk コンソールで Node.js 環境を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

コンテナオプション

次のプラットフォーム固有のオプションを指定できます。

- [プロキシサーバー] – 環境インスタンスで使用するプロキシサーバーです。デフォルトでは、NGINX が使用されます。

ログオプション

[ログ Options] セクションには、2 つの設定があります。

- インスタンスプロファイル – アプリケーションに関連付けられた Amazon S3 バケットへのアクセス許可が付与されているインスタンスプロファイルを指定します。
- [Enable log file rotation to Amazon S3] (Amazon S3 へのログファイルのローテーションの有効化) - アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーするかどうかを指定します。

静的ファイル

パフォーマンスを向上させるために、[Static files] (静的ファイル) セクションを使用して、ウェブアプリケーション内のディレクトリセットから静的ファイル (HTML、イメージなど) を配信するようにプロキシサーバーを設定することができます。ディレクトリごとに、仮想パスをディレクトリマッピングに設定します。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接　　処理します。

設定ファイルまたは、Elastic Beanstalk コンソールを使用した静的ファイルの設定の詳細については、「the section called “静的ファイル”」を参照してください。

環境プロパティ

[環境プロパティ] セクションを使用して、アプリケーションを実行している Amazon EC2 インスタンスの環境設定を指定できます。これらの設定は、キーバリューのペアでアプリケーションに渡されます。

が実行されるNode.js環境内では AWS Elastic Beanstalk、 を実行して環境変数にアクセスできますprocess.env.ENV_VARIABLE。

```
var endpoint = process.env.API_ENDPOINT
```

Node.js プラットフォームでは、プロキシサーバーがトラフィックを渡すポートに PORT 環境変数を設定します。詳細については、「プロキシサーバーを設定します」を参照してください。

詳細については「環境変数およびその他のソフトウェア設定」を参照してください。

Amazon Linux AMI (Amazon Linux 2 より前) の Node.js 環境の設定

次のコンソールソフトウェア設定カテゴリは、Amazon Linux AMI プラットフォームバージョン (Amazon Linux 2 より前) を使用する Elastic Beanstalk Node.js 環境でのみサポートされます。

> **メモ**
>
> - このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
> - 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

コンテナオプション — Amazon Linux AMI (AL1)

設定ページで、次のように指定します。

- [Proxy server] (プロキシサーバー) – Node.js へのプロキシ接続に使用するウェブサーバーを指定します。デフォルトでは、NGINX が使用されます。[none] (なし) を選択した場合、静的なファイルマッピングは有効にならず、GZIP 圧縮は無効になります。
- [Node.js version] — Node.js のバージョンを指定します。サポートされている Node.js バージョンのリストについては、「AWS Elastic Beanstalk プラットフォームガイド」の「Node.js」を参照してください。

- [GZIP compression] - GZIP 圧縮を有効にするかどうかを指定します。デフォルトでは、GZIP 圧縮は有効になっています。
- [Node command] (ノードコマンド) - Node.js アプリケーションのスタートに使用するコマンドを入力できます。空の文字列 (デフォルト) は、Elastic Beanstalk が `app.js`、`server.js`、`npm start` をこの順に使用していることを意味します。

Node.js 設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクをパフォーマンスできます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

`aws:elasticbeanstalk:environment:proxy` 名前空間を使用して、環境のインスタンスで使用するプロキシを選択できます。次の例では、Apache HTTPD プロキシサーバーを使用するように環境を設定します。

Example .ebextensions/nodejs-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
```

`aws:elasticbeanstalk:environment:proxy:staticfiles` 名前空間を使用して、静的ファイルを配信するようにプロキシを設定できます。詳細と例については、「the section called “静的ファイル”」を参照してください。

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

Amazon Linux AMI (Amazon Linux 2 より前) の Node.js プラットフォーム

Elastic Beanstalk Node.js 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションで具体的な設定と推奨事項を検討してください。

メモ

- このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
- 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

Node.js プラットフォーム固有の設定オプション — Amazon Linux AMI (AL1)

Elastic Beanstalk では、Amazon Linux AMI Node.js プラットフォームバージョンのプラットフォームに固有の設定オプションをいくつかサポートしています。アプリケーションの前に実行するプロキシサーバー、実行する Node.js の特定のバージョン、およびアプリケーションの実行に使用するコマンドを選択できます。

プロキシサーバーについては、NGINX または Apache プロキシサーバーを使用できます。none の値を `ProxyServer` オプションに設定できます。この設定では、Elastic Beanstalk は、プロキシサーバーの背後で実行されるのではなく、スタンドアロンとしてアプリケーションを実行します。環境でスタンドアロンアプリケーションを実行している場合は、NGINX でトラフィックを転送するポートをリッスンするようにコードを更新します。

```
var port = process.env.PORT || 8080;

app.listen(port, function() {
  console.log('Server running at http://127.0.0.1:%s', port);
});
```

Node.js 言語バージョン — Amazon Linux AMI (AL1)

サポートされている言語バージョンに関しては、Node.js Amazon Linux AMI プラットフォームは他の Elastic Beanstalk マネージド型プラットフォームとは異なります。これは、Node.js プラットフォームの各バージョンでサポートされている Node.js 言語バージョンが少ないためです。サポートされている Node.js バージョンのリストについては、「AWS Elastic Beanstalk プラットフォームガイド」の「Node.js」を参照してください。

プラットフォーム固有の設定オプションを使用して、言語バージョンを設定できます。手順については、「the section called “Node.js 環境の設定”」を参照してください。または、Elastic Beanstalk コンソールを使用し、プラットフォームバージョンの更新の一環として、環境で使用する Node.js のバージョンを更新します。

> **Note**
>
> 使用しているバージョンの Node.js に対する support がプラットフォームバージョンから削除された場合は、プラットフォームの更新に先立って、バージョニング設定を変更または削除する必要があります。これは、1 つ以上のバージョンの Node.js でセキュリティの脆弱性が検出された場合に発生することがあります
> この場合、設定した NodeVersion をサポートしない新しいバージョンのプラットフォームにアップグレードしようとすると、失敗します。新しい環境の作成を回避するには、古いプラットフォームバージョンと新しいプラットフォームバージョンの両方でsupport されている Node.js バージョンに NodeVersion 設定オプションを変更するか、オプション設定を削除してから、プラットフォームの更新をパフォーマンスします。

Elastic Beanstalk コンソールで環境の Node.js バージョンを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページの [プラットフォーム] で、[変更] を選択します。
4. [Update platform version] (プラットフォームのバージョンの更新) ダイアログボックスで、Node.js バージョンを選択します。
5. [保存] を選択します。

Node.js 設定の名前空間 — Amazon Linux AMI (AL1)

Node.js Amazon Linux AMI プラットフォームでは、名前空間 (`aws:elasticbeanstalk:container:nodejs:staticfiles` および `aws:elasticbeanstalk:container:nodejs`) の追加のオプションを定義します。

次の設定ファイルは、Elastic Beanstalk がアプリケーションの実行に `npm start` を使用するよう指示します。また、プロキシタイプを Apache に設定し、圧縮を有効にします。最後に、2 つのソースディレクトリから静的ファイルを提供するようにプロキシを設定します。1 つのソース

は、statichtml ソースディレクトリからのウェブサイトのルート下の html パスにある HTML ファイルです。もう 1 つのソースは、staticimages ソースディレクトリからのウェブサイトのルートの下の images パスにあるイメージファイルです。

Example .ebextensions/ノード-settings.config

```
option_settings:
  aws:elasticbeanstalk:container:nodejs:
    NodeCommand: "npm start"
    ProxyServer: apache
    GzipCompression: true
  aws:elasticbeanstalk:container:nodejs:staticfiles:
    /html: statichtml
    /images: staticimages
```

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

Elastic Beanstalk での Procfile を使用したカスタムスタートコマンドの設定

アプリケーションを起動するコマンドを指定するには、ソースバンドルのルートに Procfile というファイルを含めます。

Example Procfile

```
web: node index.js
```

Procfile の使用方法については、「ビルドファイルと Procfile」を参照してください。

> **Note**
>
> この機能により、aws:elasticbeanstalk:container:nodejs 名前空間にあるレガシーオプション NodeCommand が置き換えられます。

Elastic Beanstalk でのアプリケーションの依存関係の設定

アプリケーションには、`require()` ステートメントで指定したモジュールなど、いくつかの Node.js モジュールに対する依存関係がある可能性があります。これらのモジュールは `node_modules` ディレクトリに保存されます。アプリケーションが実行されると、Node.js でこのディレクトリからモジュールを読み込みます。詳細については、Node.js ドキュメントの「node_modules フォルダからの読み込み」を参照してください。

これらのモジュールの依存関係は、`package.json` ファイルを使用して指定できます。Elastic Beanstalk がこのファイルを検出し、`node_modules` ディレクトリが存在しない場合、Elastic Beanstalk は webapp ユーザーとして `npm install` を実行します。この `npm install` コマンドでは、Elastic Beanstalk であらかじめ作成した `node_modules` ディレクトリに依存関係をインストールします。この `npm install` コマンドでは、パブリック npm レジストリまたは他のロケーションから `package.json` ファイルにリストされているパッケージにアクセスします。詳細については、npm ドキュメントのウェブサイトを参照してください。

Elastic Beanstalk で `node_modules` ディレクトリを検出した場合、`package.json` ファイルが存在していても Elastic Beanstalk で `npm install` を実行することはありません。Elastic Beanstalk では、依存関係パッケージは `node_modules` ディレクトリ内において Node.js でアクセスして読み込めることを前提としています。

次のセクションでは、アプリケーションにおける Node.js モジュールの依存関係の確立に関する詳細を確認できます。

> **Note**
>
> Elastic Beanstalk で `npm install` の実行中にデプロイに関する問題が発生した場合は、別のアプローチを検討してください。アプリケーションソースバンドルに、依存関係モジュールを有する `node_modules` ディレクトリを含めます。そうすることで、問題を調査している間に、パブリック npm レジストリからの依存関係のインストールに伴う問題を回避できます。依存関係モジュールはローカルディレクトリから供給されるため、これを行うとデプロイ時間を短縮できる可能性もあります。詳細については、node_modules ディレクトリに Node.js の依存関係を含めるを参照してください。

package.json ファイルを使用した Node.js の依存関係の指定

プロジェクトの出典のルートに `package.json` ファイルを含めることにより、依存関係パッケージを指定し、スタートコマンドを指定します。`package.json` ファイルが存在し、プロジェ

クトソースのルートに node_modules ディレクトリが存在しない場合、Elastic Beanstalk は webapp ユーザーとして npm install を実行し、パブリック npm レジストリから依存関係をインストールします。また、Elastic Beanstalk で start コマンドを使用してアプリケーションを開始します。package.json ファイルの詳細については、npm ドキュメントのウェブサイトで「package.json ファイルでの依存関係の指定」を参照してください。

scripts キーワードを使用して、スタートコマンドを指定します。現在は、aws:elasticbeanstalk:container:nodejs 名前空間の古い NodeCommand オプションの代わりに scripts キーワードが使用されています。

Example package.json – Express

```
{
    "name": "my-app",
    "version": "0.0.1",
    "private": true,
    "dependencies": {
      "ejs": "latest",
      "aws-sdk": "latest",
      "express": "latest",
      "body-parser": "latest"
    },
    "scripts": {
      "start": "node app.js"
    }
  }
```

本番モードと開発の依存関係

package.json ファイル内で依存関係を指定するには、dependencies 属性と devDependencies 属性を使用します。dependencies 属性は、本番環境のアプリケーションに必要なパッケージを指定します。devDependencies 属性は、ローカルの開発とテストにのみ必要なパッケージを指定します。

Elastic Beanstalk は、次のコマンドを使用して webapp ユーザーとして npm install を実行します。コマンドオプションは、アプリケーションが実行されるプラットフォームブランチに含まれる npm バージョンによって異なります。

- npm v6 – Elastic Beanstalk は、デフォルトで依存関係を本番モードでインストールします。コマンド npm install --production を使用します。

- npm v7 以降 – Elastic Beanstalk は devDependencies を省略します。コマンド `npm install --omit=dev` を使用します。

上記の両方のコマンドは、devDependencies であるパッケージをインストールしません。

devDependencies パッケージをインストールする必要がある場合は、NPM_USE_PRODUCTION 環境プロパティを `false` に設定します。この設定では、npm install の実行時に上記のオプションを使用しません。これにより、devDependencies パッケージがインストールされます。

SSH と HTTPS

2023 年 3 月 7 日の Amazon Linux 2 プラットフォームリリース以降、SSH および HTTPS プロトコルを使用して Git リポジトリからパッケージを取得することもできます。プラットフォームブランチ Node.js 16 は、SSH プロトコルと HTTPS プロトコルの両方をサポートします。Node.js 14 は HTTPS プロトコルのみをサポートします。

Example package.json – Node.js 16 は HTTPS と SSH の両方をサポートします

```
...
"dependencies": {
  "aws-sdk": "https://github.com/aws/aws-sdk-js.git",
  "aws-chime": "git+ssh://git@github.com:aws/amazon-chime-sdk-js.git"
}
```

バージョンとバージョン範囲

> **Important**
>
> バージョン範囲を指定する機能は、AL2023 上で動作する Node.js プラットフォームブランチでは使用できません。AL2023 上の特定の Node.js ブランチ内では 1 つの Node.js バージョンのみがサポートされています。`package.json` ファイルでバージョン範囲が指定されている場合、その指定は無視され、Node.js のプラットフォームブランチバージョンがデフォルトで使用されます。

`package.json` ファイルで `engines` キーワードを使用して、アプリケーションで使用する Node.js のバージョンを指定します。npm 表記を使用してバージョニング範囲を指定することもできます。バージョン範囲の構文の詳細については、Node.js のウェブサイトの「npm を使用したセマンティッ

クバージョニング」を参照してください。Node.js `package.json` ファイルの `engines` キーワードでは、`aws:elasticbeanstalk:container:nodejs` 名前空間の古い `NodeVersion` オプションを置換します。

Example **`package.json`** — 単一の Node.js バージョン

```
{
    ...
    "engines": { "node" : "14.16.0" }
  }
```

Example **`package.json`** — Node.js のバージョン範囲

```
{
    ...
    "engines": { "node" : ">=10 <11" }
  }
```

バージョン範囲が指定されると、Elastic Beanstalk では、プラットフォームにあるバージョンの中から、範囲内で利用可能な最新の Node.js バージョンをインストールします。この例では、範囲は、バージョンがバージョン 10 以上で、バージョン 11 より小さくなければならないことが示されています。その結果、Elastic Beanstalk では、サポートされているプラットフォームで利用可能な最新の Node.js バージョン 10.x.y がインストールされます。

指定できるのは、プラットフォームブランチに対応する Node.js バージョンのみであることに注意してください。例えば、Node.js 16 プラットフォームブランチを使用している場合、16.x.y Node.js バージョンのみを指定できます。npm で support されているバージョン範囲オプションを使用すると、柔軟性が向上します。各プラットフォームブランチにおける有効な Node.js バージョンについては、「AWS Elastic Beanstalk プラットフォームガイド」の「Node.js」を参照してください。

> **Note**
>
> 使用しているバージョンの Node.js に対する support がプラットフォームバージョンから削除された場合は、プラットフォームの更新に先立って、Node.js バージョン設定を変更または削除する必要があります。これは、1 つ以上のバージョンの Node.js でセキュリティの脆弱性が検出された場合に発生することがあります
> この場合、設定された Node.js バージョンをsupport していないプラットフォームの新しいバージョンに更新しようとすると失敗します。新しい環境を作成する必要がないようにするには、`package.json` の Node.js のバージョニング設定を、古いプラットフォームバージョ

ンと新しいプラットフォームバージョンの両方で support されている Node.js バージョンに変更します。このトピックで前述したように、support されているバージョンを含む Node.js のバージョニング範囲を指定することもできます。また、設定を削除して、新しいソースバンドルをデプロイするオプションもあります。

node_modules ディレクトリに Node.js の依存関係を含める

アプリケーションコードとともに依存関係パッケージを環境インスタンスにデプロイするには、プロジェクトソースのルートにある `node_modules` という名前のディレクトリにそれらを含めます。詳細については、npm ドキュメントのウェブサイトで「パッケージをローカルでダウンロードしてインストールする」を参照してください。

`node_modules` ディレクトリを AL2023/AL2 Node.js プラットフォームバージョンにデプロイすると、Elastic Beanstalk では独自の依存関係パッケージが提供されていると見なして、package.json ファイルで指定された依存関係をインストールしないようにします。Node.js では `node_modules` ディレクトリで依存関係を探します。詳細については、Node.js ドキュメントの「node_modules フォルダからの読み込み」を参照してください。

> **Note**
>
> Elastic Beanstalk で `npm install` の実行中にデプロイに関する問題が発生した場合は、問題を調査している間の回避策として、このトピックで説明されているアプローチを使用することを検討してください。

Elastic Beanstalk での npm shrinkwrap による依存関係のロック

Node.js プラットフォームは、デプロイするたびに webapp ユーザーとして `npm install` を実行します。依存関係の新しいバージョンが利用可能になった場合、そのインストールがアプリケーションをデプロイする際に行われ、デプロイに長い時間がかかる可能性があります。

アプリケーションの依存関係を現在のバージョンにロックダウンする `npm-shrinkwrap.json` ファイルを作成することで、依存関係のアップグレードを回避できます。

```
$ npm install
$ npm shrinkwrap
wrote npm-shrinkwrap.json
```

依存関係が 1 回のみインストールされるようにするため、出典バンドルにこのファイルを含めます。

プロキシサーバーを設定します

Elastic Beanstalk では、リバースプロキシとして NGINX または Apache HTTPD を使用して、ポート 80 の Elastic Load Balancing ロードバランサーにアプリケーションをマッピングできます。デフォルトは NGINX です。Elastic Beanstalk にはデフォルトのプロキシ設定が用意されています。これは拡張することも、独自の設定で完全に上書きすることもできます。

デフォルトでは、Elastic Beanstalk はポート 5000 でアプリケーションにリクエストを送信するようプロキシを設定します。デフォルトのポートを上書きするには、`PORT` 環境プロパティを、主要なアプリケーションがリッスンするポートに設定します。

> **Note**
>
> アプリケーションでリッスンしているポートは、ロードバランサーからリクエストを受信するために NGINX サーバーでリッスンするポートに影響を与えません。

ご使用のプラットフォームバージョンでプロキシサーバーを設定する

すべての AL2023/AL2 プラットフォームでは、統一されたプロキシ設定機能がサポートされています。AL2023/AL2 を実行中のプラットフォームバージョンでプロキシサーバーを設定する方法の詳細については、「リバースプロキシの設定」を参照してください。

(Amazon Linux 2 より前の) Amazon Linux AMI でのプロキシの設定

Elastic Beanstalk Node.js 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションの情報を読んでください。

> **メモ**
>
> - このトピックの情報は、Amazon Linux AMI (AL1) に基づくプラットフォームブランチにのみ適用されます。AL2023/AL2 プラットフォームブランチでは、以前の Amazon Linux AMI (AL1) プラットフォームバージョンと互換性がなく、別の構成設定が必要です。
> - 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、

「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

デフォルトのプロキシ設定の拡張および上書き — Amazon Linux AMI (AL1)

Node.js プラットフォームではリバースプロキシを使用して、インスタンスのポート 80 から、ポート 8081 でリッスンしているアプリケーションにリクエストを中継します。Elastic Beanstalk にはデフォルトのプロキシ設定が用意されています。これは拡張することも、独自の設定で完全に上書きすることもできます。

デフォルト設定を拡張するには、設定ファイルで `.conf` に `/etc/nginx/conf.d` ファイルを追加します。具体的な例については、「Node.js を実行している EC2 インスタンスで HTTPS を終了する」を参照してください。

Node.js プラットフォームでは、プロキシサーバーがトラフィックを渡すポートに PORT 環境変数を設定します。コードでこの変数を読み取って、アプリケーションのポートを設定します。

```
var port = process.env.PORT || 3000;

var server = app.listen(port, function () {
    console.log('Server running at http://127.0.0.1:' + port + '/');
});
```

デフォルトの NGINX 設定では、`127.0.0.1:8081` にある `nodejs` という名前のアップストリームサーバーにトラフィックを転送します。デフォルト設定を削除し、設定ファイルに独自の設定を指定することができます。

Example .ebextensions/proxy.config

次の例では、デフォルト設定を削除し、ポート 8081 ではなく 5000 にトラフィックを転送するカスタム設定を追加します。

```
files:
  /etc/nginx/conf.d/proxy.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      upstream nodejs {
        server 127.0.0.1:5000;
```

```
        keepalive 256;
      }

      server {
        listen 8080;

        if ($time_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2})") {
            set $year $1;
            set $month $2;
            set $day $3;
            set $hour $4;
        }
        access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour
 healthd;
        access_log  /var/log/nginx/access.log  main;

        location / {
            proxy_pass  http://nodejs;
            proxy_set_header   Connection "";
            proxy_http_version 1.1;
            proxy_set_header        Host            $host;
            proxy_set_header        X-Real-IP       $remote_addr;
            proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        }

        gzip on;
        gzip_comp_level 4;
        gzip_types text/html text/plain text/css application/json application/x-
javascript text/xml application/xml application/xml+rss text/javascript;

        location /static {
            alias /var/app/current/static;
        }

      }

  /opt/elasticbeanstalk/hooks/configdeploy/post/99_kill_default_nginx.sh:
    mode: "000755"
    owner: root
    group: root
    content: |
      #!/bin/bash -xe
      rm -f /etc/nginx/conf.d/00_elastic_beanstalk_proxy.conf
      service nginx stop
```

```
      service nginx start

container_commands:
  removeconfig:
    command: "rm -f /tmp/deployment/config/
#etc#nginx#conf.d#00_elastic_beanstalk_proxy.conf /etc/nginx/
conf.d/00_elastic_beanstalk_proxy.conf"
```

設定例 (`/etc/nginx/conf.d/proxy.conf`) では、`/etc/nginx/conf.d/00_elastic_beanstalk_proxy.conf` のデフォルト設定をベースとして使用し、圧縮とログの設定および静的なファイルマッピングでデフォルトのサーバーブロックを含めます。

`removeconfig` コマンドでは、コンテナのデフォルト設定を削除し、プロキシサーバーでカスタム設定が使用されるようにします。Elastic Beanstalk は、各構成がデプロイされるときにデフォルト設定を再作成します。これを考慮して、次の例では、設定のデプロイ後のフック (`/opt/elasticbeanstalk/hooks/configdeploy/post/99_kill_default_nginx.sh`) が追加されます。これにより、デフォルト設定が削除され、プロキシサーバーが再起動されます。

> **Note**
>
> デフォルト設定は、Node.js プラットフォームの今後のバージョンで変更される可能性があります。カスタマイズのベースとして、最新バージョンの設定を使用して互換性を確保します。

デフォルト設定を上書きする場合は、静的なファイルマッピングと GZIP 圧縮を定義する必要があります。これは、プラットフォームが標準設定を適用できないためです。

Node.js のその他の Elastic Beanstalk サンプルアプリケーションとチュートリアル

このセクションでは、その他のアプリケーションとチュートリアルについて説明します。このトピックで前述した Node.js の QuickStart トピックでは、EB CLI を使用してサンプル Node.js アプリケーションを起動する方法について説明します。

Node.js アプリケーションの使用を開始するには AWS Elastic Beanstalk、最初のアプリケーションバージョンとしてアップロードし、環境にデプロイするアプリケーションソースバンドルが必要です。

Node.js サンプルアプリケーションでの環境の起動

Elastic Beanstalk は、各プラットフォームの単一ページのサンプルアプリケーションと、Amazon RDS や言語、プラットフォーム固有の機能や API などの追加 AWS リソースの使用を示すより複雑な例を提供します。 APIs

> **Note**
>
> ソースバンドル `README.md` ファイルのステップに従ってデプロイします。

サンプル

環境タイプ	出典バンドル	説明書
ウェブ・サーバー	nodejs.zip	1 ページのアプリケーション。 EB CLI でサンプルアプリケーションを起動するには、「Node.js の QuickStart」を参照してください。 Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。
Amazon RDS を搭載したウェブ・サーバー	nodejs-example-express-rds.zip	Express フレームワークと Amazon Relational Database Service (RDS) を使用するハイキングログアプリケーション。 チュートリアル
Amazon ElastiCache を搭載したウェブ・サーバー	nodejs-example-express-elasticache.zip	クラスター用の Amazon ElastiCache を使用する Express ウェブアプリケーション。クラスターはウェブアプリケーションの高可用性、パフォーマンスおよびセキュリティを拡張します。 チュートリアル

環境タイプ	出典バンドル	説明書	
DynamoDB、Amazon SNS、Amazon SQS を搭載したウェブ・サーバー	nodejs-example-dynamo.zip	新しい会社のマーケティングキャンペーンに関するユーザーのお問い合わせ情報を収集する Express ウェブサイト。 AWS SDK for JavaScript in Node.js を使用して DynamoDB テーブルにエントリを書き込み、Elastic Beanstalk 設定ファイルを使用して DynamoDB、Amazon SNS、Amazon SQS にリソースを作成します。 チュートリアル	

次のステップ

環境でアプリケーションを実行すると、アプリケーションの新しいバージョンや、まったく異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。アプリケーションのデプロイの詳細については、「アプリケーションの新しいバージョニングをデプロイする」を参照してください。

サンプルアプリケーションを 1 つか 2 つデプロイし、Node.js アプリケーションをローカルで開発および実行をスタートする準備が整ったら、「Elastic Beanstalk 用の Node.js 開発環境の設定」を参照し、必要なすべてのツールとともに Node.js 開発環境を設定します。

Node.js Express アプリケーションの Elastic Beanstalk へのデプロイ

このセクションでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用して Elastic Beanstalk にサンプルアプリケーションをデプロイした後、Express フレームワークを使用するようにアプリケーションを更新する手順を説明します。

前提条件

このチュートリアルでは、次の前提条件が必要です。

- Node.js ランタイム

- デフォルトの Node.js パッケージマネージャーソフトウェア、npm
- Express コマンドラインジェネレーター
- Elastic Beanstalk コマンドラインインターフェイス (EB CLI)

最初の 3 つのリストされたコンポーネントのインストールとローカル開発環境の設定の詳細については、「Elastic Beanstalk 用の Node.js 開発環境の設定」を参照してください。このチュートリアルでは、 AWS SDK for Node.js をインストールする必要はありません。これは、参照されるトピックでも説明されています。

EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

Elastic Beanstalk 環境の作成

アプリケーションディレクトリ

このチュートリアルでは、アプリケーションソースバンドル用に `nodejs-example-express-rds` と呼ばれるディレクトリを使用します。このチュートリアル用の `nodejs-example-express-rds` ディレクトリを作成します。

```
~$ mkdir nodejs-example-express-rds
```

> **Note**
> この章の各チュートリアルでは、アプリケーションソースバンドル用に独自のディレクトリを使用します。ディレクトリ名は、チュートリアルで使用されるサンプルアプリケーションの名前と一致します。

現在の作業ディレクトリを `nodejs-example-express-rds` に変更します。

```
~$ cd nodejs-example-express-rds
```

次に、Node.js プラットフォームとサンプルアプリケーションを実行する Elastic Beanstalk 環境を設定しましょう。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。

アプリケーションの EB CLI リポジトリを設定し、Node.js プラットフォームを実行する Elastic Beanstalk 環境を作成するには

1. eb init コマンドを使用してリポジトリを作成します。

```
~/nodejs-example-express-rds$ eb init --platform node.js --region <region>
```

このコマンドは、`.elasticbeanstalk` という名前のフォルダに、アプリケーションの環境作成用の設定ファイルを作成し、現在のフォルダに基づいた名前で Elastic Beanstalk アプリケーションを作成します。

2. eb create コマンドを使用して、サンプルアプリケーションを実行する環境を作成します。

```
~/nodejs-example-express-rds$ eb create --sample nodejs-example-express-rds
```

このコマンドは、Node.js プラットフォームと以下のリソース用にデフォルト設定でロードバランスされた環境を作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブ・アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。

- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。

- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。
> Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

3. 環境の作成が完了したら、eb open コマンドを使用して、デフォルトのブラウザでその環境の URL を開きます。

```
~/nodejs-example-express-rds$ eb open
```

これで、サンプルアプリケーションを使用して Node.js Elastic Beanstalk 環境が作成されました。独自のアプリケーションで更新できます。次に、Express フレームワークを使用するようサンプルアプリケーションを更新します。

Express を使用するようアプリケーションを更新する

サンプルアプリケーションの環境を作成したら、その環境を使用するようにアプリケーションを更新できます。この手順では、まず express と npm install コマンドを実行して、アプリケーションディレクトリで Express フレームワークを設定します。その後、EB CLI を使用して、更新されたアプリケーションで Elastic Beanstalk 環境を更新します。

Express を使用するようアプリケーションを更新するには

1. `express` コマンドを実行します。これによって、`package.json` と `app.js`、およびいくつかのディレクトリが生成されます。

   ```
   ~/nodejs-example-express-rds$ express
   ```

   プロンプトが表示されたら、続行するには **y** と入力します。

   > **Note**
   > express コマンドが機能しない場合は、前述の「前提条件」セクションで説明したように Express コマンドラインジェネレーターがインストールされていない可能性があります。または、express コマンドを実行するために、ローカルマシンのディレクトリパス設定を行う必要がある場合があります。開発環境の設定に関する詳細なステップについては、「前提条件」セクションを参照して、このチュートリアルを続行してください。

2. ローカルの依存関係を設定します。

   ```
   ~/nodejs-example-express-rds$ npm install
   ```

3. (オプション) ウェブアプリサーバーが起動することを確認します。

   ```
   ~/nodejs-example-express-rds$ npm start
   ```

   次のような出力が表示されます:

   ```
   > nodejs@0.0.0 start /home/local/user/node-express
   > node ./bin/www
   ```

サーバーは、デフォルトでポート 3000 で実行されます。テストするには、別のターミナルで `curl http://localhost:3000` を実行するか、ローカルコンピュータでブラウザを開いて URL アドレス `http://localhost:3000` を入力します。

サーバーを停止するには、[Ctrl+C] を押します。

4. eb deploy コマンドを使用して変更を Elastic Beanstalk 環境にデプロイします。

```
~/nodejs-example-express-rds$ eb deploy
```

5. 環境が緑色で示されていて準備完了したら、URL を再表示して正しく動作することを確認します。ウェブ・ページに "[Welcome to Express]" が表示されます。

次に、静的ファイルを処理し、新しいページを追加するように Express アプリケーションを更新します。

静的ファイルを設定し、新しいページを Express アプリケーションに追加します。

1. 次の内容を含む 2 つ目の設定ファイルを `.ebextensions` フォルダに追加します。

   `nodejs-example-express-rds/.ebextensions/staticfiles.config`

```
option_settings:
    aws:elasticbeanstalk:environment:proxy:staticfiles:
        /stylesheets: public/stylesheets
```

   この設定では、プロキシサーバーに `public` フォルダのファイルを、アプリケーションの `/public` のパスで提供するように設定します。プロキシサーバーから静的にファイルを提供すると、アプリケーションへのロードを減らすことができます。詳細については、この章の前半の「静的ファイル」を参照してください。

2. (オプション) 静的マッピングが正しく設定されていることを確認するには、`nodejs-example-express-rds/app.js` の静的マッピング設定をコメントアウトします。これにより、ノードアプリケーションからマッピングが削除されます。

```
//  app.use(express.static(path.join(__dirname, 'public')));
```

   この行をコメントアウトした後でも、前のステップの `staticfiles.config` ファイル内の静的ファイルマッピングは、スタイルシートを引き続き正常にロードするはずです。静的ファイル

マッピングが Express アプリケーションではなくプロキシ静的ファイル設定を通じてロードされていることを確認するには、`option_settings:` の後の値を削除します。静的ファイル設定とノードアプリケーションの両方からその値が削除されると、スタイルシートはロードできなくなります。

テストが完了したら、`nodejs-example-express-rds/app.js` と `staticfiles.config` の両方の内容を忘れずにリセットしてください。

3. `nodejs-example-express-rds/routes/hike.js` を追加します。次の内容を入力します。

```
exports.index = function(req, res) {
 res.render('hike', {title: 'My Hiking Log'});
};

exports.add_hike = function(req, res) {
};
```

4. `nodejs-example-express-rds/app.js` を更新して新しく 3 つの行を含めます。

   まず、次の行を追加して、このルートに `require` を追加します。

```
var hike = require('./routes/hike');
```

   ファイルは次のスニペットのようになります。

```
var express = require('express');
var path = require('path');
var hike = require('./routes/hike');
```

   その後、次の 2 つの行を `nodejs-example-express-rds/app.js` の `var app = express();` の後に追加します。

```
app.get('/hikes', hike.index);
app.post('/add_hike', hike.add_hike);
```

   ファイルは次のスニペットのようになります。

```
var app = express();
app.get('/hikes', hike.index);
app.post('/add_hike', hike.add_hike);
```

5. `nodejs-example-express-rds/views/index.jade` を `nodejs-example-express-rds/views/hike.jade` にコピーします。

```
~/nodejs-example-express-rds$ cp views/index.jade views/hike.jade
```

6. eb deploy コマンドを使用して変更をデプロイします。

```
~/nodejs-example-express-rds$ eb deploy
```

7. 数分後、環境が更新されます。環境が緑色で示されていて準備が完了したら、ブラウザを再表示し、URL の最後に **hikes** を追加して (`http://node-express-env-syypntcz2q.elasticbeanstalk.com/hikes` など)、正しく動作することを確認します。

   タイトルが [My Hiking ログ] のウェブ・ページが表示されます。

これで、Express フレームワークを使用するウェブアプリケーションが作成されました。次のセクションでは、Amazon Relational Database Service (RDS) を使用してハイキングログを保存するようにアプリケーションを変更します。

Amazon RDS を使用するようにアプリケーションを更新する

この次のステップでは、Amazon RDS for MySQL を使用するようにアプリケーションを更新します。

RDS for MySQL を使用するようにアプリケーションを更新するには

1. Elastic Beanstalk 環境に結合された RDS for MySQL データベースを作成するには、この章で後述する「データベースの追加」トピックの手順に従ってください。データベースインスタンスの追加には約 10 分かかります。
2. `package.json` の依存関係セクションを次の内容で更新します。

```
"dependencies": {
    "async": "^3.2.4",
    "express": "4.18.2",
    "jade": "1.11.0",
    "mysql": "2.18.1",
    "node-uuid": "^1.4.8",
    "body-parser": "^1.20.1",
    "method-override": "^3.0.0",
    "morgan": "^1.10.0",
```

```
    "errorhandler": "^1.5.1"
  }
```

3. npm install を実行します。

```
~/nodejs-example-express-rds$ npm install
```

4. app.js を更新してデータベースに接続し、テーブルを作成し、単一のデフォルトのハイキングログを挿入します。このアプリがデプロイされるたびに、前の hikes テーブルが削除され、再作成されます。

```
/**
 * Module dependencies.
 */

 const express = require('express')
 , routes = require('./routes')
 , hike = require('./routes/hike')
 , http = require('http')
 , path = require('path')
 , mysql = require('mysql')
 , async = require('async')
 , bodyParser = require('body-parser')
 , methodOverride = require('method-override')
 , morgan = require('morgan')
 , errorhandler = require('errorhandler');

const { connect } = require('http2');

const app = express()

app.set('views', __dirname + '/views')
app.set('view engine', 'jade')
app.use(methodOverride())
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({ extended: true }))
app.use(express.static(path.join(__dirname, 'public')))


app.set('connection', mysql.createConnection({
host: process.env.RDS_HOSTNAME,
user: process.env.RDS_USERNAME,
password: process.env.RDS_PASSWORD,
```

```
port: process.env.RDS_PORT}));

function init() {
 app.get('/', routes.index);
 app.get('/hikes', hike.index);
 app.post('/add_hike', hike.add_hike);
}

const client = app.get('connection');
async.series([
 function connect(callback) {
   client.connect(callback);
   console.log('Connected!');
 },
 function clear(callback) {
   client.query('DROP DATABASE IF EXISTS mynode_db', callback);
 },
 function create_db(callback) {
   client.query('CREATE DATABASE mynode_db', callback);
 },
 function use_db(callback) {
   client.query('USE mynode_db', callback);
 },
 function create_table(callback) {
    client.query('CREATE TABLE HIKES (' +
                        'ID VARCHAR(40), ' +
                        'HIKE_DATE DATE, ' +
                        'NAME VARCHAR(40), ' +
                        'DISTANCE VARCHAR(40), ' +
                        'LOCATION VARCHAR(40), ' +
                        'WEATHER VARCHAR(40), ' +
                        'PRIMARY KEY(ID))', callback);
 },
 function insert_default(callback) {
   const hike = {HIKE_DATE: new Date(), NAME: 'Hazard Stevens',
         LOCATION: 'Mt Rainier', DISTANCE: '4,027m vertical', WEATHER:'Bad', ID:
 '12345'};
   client.query('INSERT INTO HIKES set ?', hike, callback);
 }
], function (err, results) {
 if (err) {
   console.log('Exception initializing database.');
   throw err;
 } else {
```

```
   console.log('Database initialization complete.');
   init();
 }
});

module.exports = app
```

5. `routes/hike.js` に次の内容を追加します。これにより、ルートは新しいハイキングログを HIKES データベースに挿入できるようになります。

```
const uuid = require('node-uuid');
exports.index = function(req, res) {
  res.app.get('connection').query( 'SELECT * FROM HIKES', function(err,
rows) {
    if (err) {
      res.send(err);
    } else {
      console.log(JSON.stringify(rows));
      res.render('hike', {title: 'My Hiking Log', hikes: rows});
  }});
};
exports.add_hike = function(req, res){
  const input = req.body.hike;
  const hike = { HIKE_DATE: new Date(), ID: uuid.v4(), NAME: input.NAME,
  LOCATION: input.LOCATION, DISTANCE: input.DISTANCE, WEATHER: input.WEATHER};
  console.log('Request to log hike:' + JSON.stringify(hike));
  req.app.get('connection').query('INSERT INTO HIKES set ?', hike, function(err) {
      if (err) {
        res.send(err);
      } else {
        res.redirect('/hikes');
      }
   });
};
```

6. `routes/index.js` のコンテンツを次と置き換えます。

```
/*
 * GET home page.
 */

exports.index = function(req, res){
  res.render('index', { title: 'Express' });
```

```
};
```

7. 次の jade テンプレートを `views/hike.jade` に追加して、ハイキングログを追加するためのユーザーインターフェイスを指定します。

```
extends layout

block content
  h1= title
  p Welcome to #{title}

  form(action="/add_hike", method="post")
    table(border="1")
      tr
        td Your Name
        td
          input(name="hike[NAME]", type="textbox")
      tr
        td Location
        td
          input(name="hike[LOCATION]", type="textbox")
      tr
        td Distance
        td
          input(name="hike[DISTANCE]", type="textbox")
      tr
        td Weather
        td
          input(name="hike[WEATHER]", type="radio", value="Good")
          | Good
          input(name="hike[WEATHER]", type="radio", value="Bad")
          | Bad
          input(name="hike[WEATHER]", type="radio", value="Seattle", checked)
          | Seattle
      tr
        td(colspan="2")
          input(type="submit", value="Record Hike")

  div
    h3 Hikes
    table(border="1")
      tr
        td Date
```

```
        td Name
        td Location
        td Distance
        td Weather
      each hike in hikes
        tr
          td #{hike.HIKE_DATE.toDateString()}
          td #{hike.NAME}
          td #{hike.LOCATION}
          td #{hike.DISTANCE}
          td #{hike.WEATHER}
```

8. eb deploy コマンドを使用して変更をデプロイします。

```
~/nodejs-example-express-rds$ eb deploy
```

クリーンアップ

Elastic Beanstalk での作業が終了したら、環境を終了できます。

eb terminate コマンドを使用して、お客様の環境とその環境に含まれるすべてのリソースを終了します。

```
~/nodejs-example-express-rds$ eb terminate
The environment "nodejs-example-express-rds-env" and all associated instances will be
 terminated.
To confirm, type the environment name: nodejs-example-express-rds-env
INFO: terminateEnvironment is starting.
...
```

クラスタリング付き Node.js Express アプリケーションの Elastic Beanstalk へのデプロイ

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用してサンプルアプリケーションを Elastic Beanstalk にデプロイした後、Express フレームワーク、Amazon ElastiCache、およびクラスタリングを使用するようアプリケーションを更新する手順を示します。クラスターはウェブアプリケーションの高可用性、パフォーマンス、セキュリティを拡張します。Amazon ElastiCache の詳細については、「Amazon ElastiCache (Memcached) ユーザーガイド」の「Amazon ElastiCache (Memcached) とは」を参照してください。

> **Note**
>
> この例では、課金される可能性のある AWS リソースを作成します。 AWS 料金の詳細については、「」を参照してくださいhttps://aws.amazon.com/pricing/。一部の サービスは AWS 無料利用枠の一部です。新規のお客様は、無料でこれらのサービスをテストできる場合があります。詳細については「https://aws.amazon.com/free/」を参照してください。

前提条件

このチュートリアルでは、次の前提条件が必要です。

- Node.js ランタイム
- デフォルトの Node.js パッケージマネージャーソフトウェア、npm
- Express コマンドラインジェネレーター
- Elastic Beanstalk コマンドラインインターフェイス (EB CLI)

最初の 3 つのリストされたコンポーネントのインストールとローカル開発環境の設定の詳細については、「Elastic Beanstalk 用の Node.js 開発環境の設定」を参照してください。このチュートリアルでは、 AWS SDK for Node.js をインストールする必要はありません。これは、参照されるトピックでも説明されています。

EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

Elastic Beanstalk 環境の作成

アプリケーションディレクトリ

このチュートリアルでは、アプリケーションソースバンドル用に `nodejs-example-express-elasticache` と呼ばれるディレクトリを使用します。このチュートリアル用の `nodejs-example-express-elasticache` ディレクトリを作成します。

```
~$ mkdir nodejs-example-express-elasticache
```

> **Note**
>
> この章の各チュートリアルでは、アプリケーションソースバンドル用に独自のディレクトリを使用します。ディレクトリ名は、チュートリアルで使用されるサンプルアプリケーションの名前と一致します。

現在の作業ディレクトリを `nodejs-example-express-elasticache` に変更します。

```
~$ cd nodejs-example-express-elasticache
```

次に、Node.js プラットフォームとサンプルアプリケーションを実行する Elastic Beanstalk 環境を設定しましょう。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。

アプリケーションの EB CLI リポジトリを設定し、Node.js プラットフォームを実行する Elastic Beanstalk 環境を作成するには

1. eb init コマンドを使用してリポジトリを作成します。

   ```
   ~/nodejs-example-express-elasticache$ eb init --platform node.js --region <region>
   ```

   このコマンドは、`.elasticbeanstalk` という名前のフォルダに、アプリケーションの環境作成用の設定ファイルを作成し、現在のフォルダに基づいた名前で Elastic Beanstalk アプリケーションを作成します。

2. eb create コマンドを使用して、サンプルアプリケーションを実行する環境を作成します。

   ```
   ~/nodejs-example-express-elasticache$ eb create --sample nodejs-example-express-elasticache
   ```

   このコマンドは、Node.js プラットフォームと以下のリソース用にデフォルト設定でロードバランスされた環境を作成します。

   - EC2 インスタンス – 選択したプラットフォームでウェブ・アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

     各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれか

をウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。

- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。
> Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの

付いた Cookie の使用をお勧めします。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御します。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。

3. 環境の作成が完了したら、eb open コマンドを使用して、デフォルトのブラウザでその環境の URL を開きます。

```
~/nodejs-example-express-elasticache$ eb open
```

これで、サンプルアプリケーションを使用して Node.js Elastic Beanstalk 環境が作成されました。独自のアプリケーションで更新できます。次に、Express フレームワークを使用するようサンプルアプリケーションを更新します。

Express を使用するようアプリケーションを更新する

Elastic Beanstalk 環境にサンプルアプリケーションを更新して、Express フレームワークを使用するようにします。

最終的な出典コードは、nodejs-example-express-elasticache.zip からダウンロードすることができます。

Express を使用するようアプリケーションを更新するには

サンプルアプリケーションの環境を作成したら、その環境を使用するようにアプリケーションを更新できます。この手順では、まず express と npm install コマンドを実行して、アプリケーションディレクトリで Express フレームワークを設定します。

1. `express` コマンドを実行します。これによって、`package.json` と `app.js`、およびいくつかのディレクトリが生成されます。

```
~/nodejs-example-express-elasticache$ express
```

プロンプトが表示されたら、続行するには **y** と入力します。

> **Note**
>
> express コマンドが機能しない場合は、前述の「前提条件」セクションで説明したように Express コマンドラインジェネレーターがインストールされていない可能性があります。または、express コマンドを実行するために、ローカルマシンのディレクトリパス

設定を行う必要がある場合があります。開発環境の設定に関する詳細なステップについては、「前提条件」セクションを参照して、このチュートリアルを続行してください。

2. ローカルの依存関係を設定します。

```
~/nodejs-example-express-elasticache$ npm install
```

3. (オプション) ウェブアプリサーバーが起動することを確認します。

```
~/nodejs-example-express-elasticache$ npm start
```

次のような出力が表示されます:

```
> nodejs@0.0.0 start /home/local/user/node-express
> node ./bin/www
```

サーバーは、デフォルトでポート 3000 で実行されます。テストするには、別のターミナルで `curl http://localhost:3000` を実行するか、ローカルコンピュータでブラウザを開いて URL アドレス `http://localhost:3000` を入力します。

サーバーを停止するには、[Ctrl+C] を押します。

4. `nodejs-example-express-elasticache/app.js` を `nodejs-example-express-elasticache/express-app.js` に名前変更します。

```
~/nodejs-example-express-elasticache$ mv app.js express-app.js
```

5. `nodejs-example-express-elasticache/express-app.js` の `var app = express();` 行を次に更新します。

```
var app = module.exports = express();
```

6. ローカルコンピュータで、以下のコードを含む `nodejs-example-express-elasticache/app.js` という名前のファイルを作成します。

```
/**
 * Module dependencies.
 */

 const express = require('express'),
```

```
 session = require('express-session'),
 bodyParser = require('body-parser'),
 methodOverride = require('method-override'),
 cookieParser = require('cookie-parser'),
 fs = require('fs'),
 filename = '/var/nodelist',
 app = express();

let MemcachedStore = require('connect-memcached')(session);

function setup(cacheNodes) {
 app.use(bodyParser.raw());
 app.use(methodOverride());
 if (cacheNodes.length > 0) {
   app.use(cookieParser());

   console.log('Using memcached store nodes:');
   console.log(cacheNodes);

   app.use(session({
     secret: 'your secret here',
     resave: false,
     saveUninitialized: false,
     store: new MemcachedStore({ 'hosts': cacheNodes })
   }));
 } else {
   console.log('Not using memcached store.');
   app.use(session({
     resave: false,
     saveUninitialized: false, secret: 'your secret here'
   }));
 }

 app.get('/', function (req, resp) {
   if (req.session.views) {
     req.session.views++
     resp.setHeader('Content-Type', 'text/html')
     resp.send(`You are session: ${req.session.id}. Views: ${req.session.views}`)
   } else {
     req.session.views = 1
     resp.send(`You are session: ${req.session.id}. No views yet, refresh the page!
`)
   }
 });
```

```
 if (!module.parent) {
   console.log('Running express without cluster. Listening on port %d',
 process.env.PORT || 5000)
   app.listen(process.env.PORT || 5000)
 }
}

console.log("Reading elastic cache configuration")
// Load elasticache configuration.
fs.readFile(filename, 'UTF8', function (err, data) {
 if (err) throw err;

 let cacheNodes = []
 if (data) {
   let lines = data.split('\n');
   for (let i = 0; i < lines.length; i++) {
     if (lines[i].length > 0) {
       cacheNodes.push(lines[i])
     }
   }
 }

 setup(cacheNodes)
});

module.exports = app;
```

7. `nodejs-example-express-elasticache/bin/www` ファイルの内容を次に置き換えます。

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

const app = require('../app');
const cluster = require('cluster');
const debug = require('debug')('nodejs-example-express-elasticache:server');
const http = require('http');
const workers = {},
  count = require('os').cpus().length;

function spawn() {
```

```
  const worker = cluster.fork();
  workers[worker.pid] = worker;
  return worker;
}


/**
 * Get port from environment and store in Express.
 */

const port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

if (cluster.isMaster) {
  for (let i = 0; i < count; i++) {
    spawn();
  }

  // If a worker dies, log it to the console and start another worker.
  cluster.on('exit', function (worker, code, signal) {
    console.log('Worker ' + worker.process.pid + ' died.');
    cluster.fork();
  });

  // Log when a worker starts listening
  cluster.on('listening', function (worker, address) {
    console.log('Worker started with PID ' + worker.process.pid + '.');
  });

} else {
  /**
   * Create HTTP server.
   */

  let server = http.createServer(app);

  /**
   * Event listener for HTTP server "error" event.
   */

  function onError(error) {
    if (error.syscall !== 'listen') {
      throw error;
    }
```

```
    const bind = typeof port === 'string'
      ? 'Pipe ' + port
      : 'Port ' + port;

    // handle specific listen errors with friendly messages
    switch (error.code) {
      case 'EACCES':
        console.error(bind + ' requires elevated privileges');
        process.exit(1);
        break;
      case 'EADDRINUSE':
        console.error(bind + ' is already in use');
        process.exit(1);
        break;
      default:
        throw error;
    }
  }

  /**
   * Event listener for HTTP server "listening" event.
   */

  function onListening() {
    const addr = server.address();
    const bind = typeof addr === 'string'
      ? 'pipe ' + addr
      : 'port ' + addr.port;
    debug('Listening on ' + bind);
  }

  /**
   * Listen on provided port, on all network interfaces.
   */

  server.listen(port);
  server.on('error', onError);
  server.on('listening', onListening);
}

/**
 * Normalize a port into a number, string, or false.
 */
```

```
function normalizePort(val) {
  const port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}
```

8. eb deploy コマンドを使用して変更を Elastic Beanstalk 環境にデプロイします。

```
~/nodejs-example-express-elasticache$ eb deploy
```

9. 数分後、環境が更新されます。環境が緑色で示されていて準備完了したら、URL を再表示して正しく動作することを確認します。ウェブページに "Welcome to Express" が表示されます。

アプリケーションを実行している EC2 インスタンスのログにアクセスできます。ログのアクセス手順については、「Elastic Beanstalk 環境の Amazon EC2 インスタンスからのログの表示」を参照してください。

次に、Amazon ElastiCache を使用するよう Express アプリケーションを更新します。

Amazon ElastiCache を使用するよう Express アプリケーションを更新するには

1. ローカルコンピュータで、出典バンドルの最上位ディレクトリに `.ebextensions` ディレクトリを作成します。この例では、`nodejs-example-express-elasticache/.ebextensions` を使用します。
2. 次のスニペットを使用して、設定ファイル `nodejs-example-express-elasticache/.ebextensions/elasticache-iam-with-script.config` を作成します。設定ファイルの詳細については、「Node.js 設定の名前空間」を参照してください。elasticache ノードの検出に必要な許可を持つ IAM ユーザーが作成され、キャッシュが変更されると常にファイルに書き込みされます。nodejs-example-express-elasticache.zip か

らファイルをコピーすることもできます。ElastiCache プロパティの詳細については、「例: ElastiCache」を参照してください。

> **Note**
>
> YAML は、一貫したインデントに依存します。設定ファイルの例でコンテンツを置き換える際はインデントレベルを一致させ、テキストエディタがインデントにタブ文字ではなくスペースを使用していることを確認します。

```
Resources:
  MyCacheSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: "Lock cache down to webserver access only"
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort:
            Fn::GetOptionSetting:
              OptionName: CachePort
              DefaultValue: 11211
          ToPort:
            Fn::GetOptionSetting:
              OptionName: CachePort
              DefaultValue: 11211
          SourceSecurityGroupName:
            Ref: AWSEBSecurityGroup
  MyElastiCache:
    Type: 'AWS::ElastiCache::CacheCluster'
    Properties:
      CacheNodeType:
        Fn::GetOptionSetting:
          OptionName: CacheNodeType
          DefaultValue: cache.t2.micro
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName: NumCacheNodes
          DefaultValue: 1
      Engine:
        Fn::GetOptionSetting:
          OptionName: Engine
          DefaultValue: redis
```

```
      VpcSecurityGroupIds:
        -
          Fn::GetAtt:
            - MyCacheSecurityGroup
            - GroupId
  AWSEBAutoScalingGroup :
    Metadata :
      ElastiCacheConfig :
        CacheName :
          Ref : MyElastiCache
        CacheSize :
           Fn::GetOptionSetting:
             OptionName : NumCacheNodes
             DefaultValue: 1
  WebServerUser :
    Type : AWS::IAM::User
    Properties :
      Path : "/"
      Policies:
        -
          PolicyName: root
          PolicyDocument :
            Statement :
              -
                Effect : Allow
                Action :
                  - cloudformation:DescribeStackResource
                  - cloudformation:ListStackResources
                  - elasticache:DescribeCacheClusters
                Resource : "*"
  WebServerKeys :
    Type : AWS::IAM::AccessKey
    Properties :
      UserName :
        Ref: WebServerUser

Outputs:
  WebsiteURL:
    Description: sample output only here to show inline string function parsing
    Value: |
      http://`{ "Fn::GetAtt" : [ "AWSEBLoadBalancer", "DNSName" ] }`
  MyElastiCacheName:
    Description: Name of the elasticache
    Value:
```

```
      Ref : MyElastiCache
  NumCacheNodes:
    Description: Number of cache nodes in MyElastiCache
    Value:
      Fn::GetOptionSetting:
        OptionName : NumCacheNodes
        DefaultValue: 1

files:
  "/etc/cfn/cfn-credentials" :
    content : |
      AWSAccessKeyId=`{ "Ref" : "WebServerKeys" }`
      AWSSecretKey=`{ "Fn::GetAtt" : ["WebServerKeys", "SecretAccessKey"] }`
    mode : "000400"
    owner : root
    group : root

  "/etc/cfn/get-cache-nodes" :
    content : |
      # Define environment variables for command line tools
      export AWS_ELASTICACHE_HOME="/home/ec2-user/elasticache/$(ls /home/ec2-user/
elasticache/)"
      export AWS_CLOUDFORMATION_HOME=/opt/aws/apitools/cfn
      export PATH=$AWS_CLOUDFORMATION_HOME/bin:$AWS_ELASTICACHE_HOME/bin:$PATH
      export AWS_CREDENTIAL_FILE=/etc/cfn/cfn-credentials
      export JAVA_HOME=/usr/lib/jvm/jre

      # Grab the Cache node names and configure the PHP page
      aws cloudformation list-stack-resources --stack `{ "Ref" :
 "AWS::StackName" }` --region `{ "Ref" : "AWS::Region" }` --output text | grep
 MyElastiCache | awk '{print $4}' | xargs -I {} aws elasticache describe-cache-
clusters --cache-cluster-id {} --region `{ "Ref" : "AWS::Region" }` --show-
cache-node-info --output text | grep '^ENDPOINT' | awk '{print $2 ":" $3}' >
 `{ "Fn::GetOptionSetting" : { "OptionName" : "NodeListPath", "DefaultValue" : "/
var/www/html/nodelist" } }`
    mode : "000500"
    owner : root
    group : root

  "/etc/cfn/hooks.d/cfn-cache-change.conf" :
    "content": |
      [cfn-cache-size-change]
      triggers=post.update
      path=Resources.AWSEBAutoScalingGroup.Metadata.ElastiCacheConfig
```

```
      action=/etc/cfn/get-cache-nodes
      runas=root

sources :
  "/home/ec2-user/elasticache" : "https://elasticache-downloads.s3.amazonaws.com/
AmazonElastiCacheCli-latest.zip"

commands:
  make-elasticache-executable:
    command: chmod -R ugo+x /home/ec2-user/elasticache/*/bin/*

packages :
  "yum" :
    "aws-apitools-cfn"  : []

container_commands:
  initial_cache_nodes:
    command: /etc/cfn/get-cache-nodes
```

3. ローカルコンピュータに次のスニペットを使用して設定ファイル `nodejs-example-express-elasticache/.ebextensions/elasticache_settings.config` を作成し、ElastiCache を設定します。

```
option_settings:
  "aws:elasticbeanstalk:customoption":
     CacheNodeType: cache.t2.micro
     NumCacheNodes: 1
     Engine: memcached
     NodeListPath: /var/nodelist
```

4. ローカルコンピュータで、`nodejs-example-express-elasticache/express-app.js` を次のスニペットに置き換えます。このファイルでは、ディスクからノードリストを読み取り (`/var/nodelist`)、ノードが存在する場合は `memcached` をセッションを保存するとして使用するよう Express を設定します。ファイルは次のようになります。

```
/**
 * Module dependencies.
 */

var express = require('express'),
    session = require('express-session'),
    bodyParser = require('body-parser'),
```

```
    methodOverride = require('method-override'),
    cookieParser = require('cookie-parser'),
    fs = require('fs'),
    filename = '/var/nodelist',
    app = module.exports = express();

var MemcachedStore = require('connect-memcached')(session);

function setup(cacheNodes) {
  app.use(bodyParser.raw());
  app.use(methodOverride());
  if (cacheNodes) {
      app.use(cookieParser());

      console.log('Using memcached store nodes:');
      console.log(cacheNodes);

      app.use(session({
          secret: 'your secret here',
          resave: false,
          saveUninitialized: false,
          store: new MemcachedStore({'hosts': cacheNodes})
      }));
  } else {
    console.log('Not using memcached store.');
    app.use(cookieParser('your secret here'));
    app.use(session());
  }

  app.get('/', function(req, resp){
  if (req.session.views) {
      req.session.views++
      resp.setHeader('Content-Type', 'text/html')
      resp.write('Views: ' + req.session.views)
      resp.end()
   } else {
      req.session.views = 1
      resp.end('Refresh the page!')
    }
  });

  if (!module.parent) {
      console.log('Running express without cluster.');
      app.listen(process.env.PORT || 5000);
```

```
  }
}

// Load elasticache configuration.
fs.readFile(filename, 'UTF8', function(err, data) {
    if (err) throw err;

    var cacheNodes = [];
    if (data) {
        var lines = data.split('\n');
        for (var i = 0 ; i < lines.length ; i++) {
            if (lines[i].length > 0) {
                cacheNodes.push(lines[i]);
            }
        }
    }
    setup(cacheNodes);
});
```

5. ローカルコンピュータで、`package.json` を次の内容で更新します。

```
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "jade": "~1.11.0",
    "morgan": "~1.9.1",
    "connect-memcached": "*",
    "express-session": "*",
    "body-parser": "*",
    "method-override": "*"
  }
```

6. npm install を実行します。

```
~/nodejs-example-express-elasticache$ npm install
```

7. 更新したアプリケーションをデプロイします。

```
~/nodejs-example-express-elasticache$ eb deploy
```

8. 数分後、環境が更新されます。環境が緑色で示され準備が完了したら、コードが機能することを確認します。

   a. Amazon CloudWatch コンソール をチェックして、 ElastiCache メトリクスを表示します。ElastiCache メトリクスを表示するには、左ペインで [Metrics (メトリクス)] を選択し、[CurrItems] を検索します。[ElastiCache> Cache Node Metrics] を選択したら、キャッシュノードを選択してキャッシュ内の項目数を表示します。

   > **Note**
   > アプリケーションのデプロイ先と同じリージョンを調べていることを確認してください。

   アプリケーション URL をコピーして別のウェブ・ブラウザに貼り付け、ページを更新した場合、5 分後に CurrItem カウントが上がります。

   b. ログのスナップショットを取得します。ログの取得の詳細については、「Elastic Beanstalk 環境の Amazon EC2 インスタンスからのログの表示」を参照してください。

   c. ログバンドルの `/var/log/nodejs/nodejs.log` ファイルをチェックします。次のような結果が表示されます。

```
Using memcached store nodes:
[ 'aws-my-1oys9co8zt1uo.1iwtrn.0001.use1.cache.amazonaws.com:11211' ]
```

クリーンアップ

アプリケーションを実行したくない場合は、環境を終了し、アプリケーションを削除してクリーンアップできます。

環境を終了するには `eb terminate` コマンドを、アプリケーションを削除するには `eb delete` コマンドを使用します。

環境を終了するには

ローカルリポジトリを作成したディレクトリから、`eb terminate` を実行します。

```
$ eb terminate
```

このプロセスには数分かかることがあります。環境が正常に終了すると、Elastic Beanstalk にメッセージが表示されます。

DynamoDB を使用して Node.js アプリケーションを Elastic Beanstalk にデプロイする

このチュートリアルとそのサンプルアプリケーション nodejs-example-dynamo.zip では、Node.js で AWS SDK for JavaScript を使用して Amazon DynamoDB サービスとやり取りする Node.js アプリケーションをデプロイするプロセスについて説明します。 AWS Elastic Beanstalk 環境から分離されたデータベース、または外部にある DynamoDB テーブルを作成します。また、分離されたデータベースを使用するようにアプリケーションを設定します。本番環境では、環境のライフサイクルから独立した状態であるように、Elastic Beanstalk 環境から分離されたデータベースを使用するのがベストプラクティスです。このプラクティスにより、ブルー/グリーンデプロイの実行も可能となります。

サンプルアプリケーションは次を説明します。

- ユーザーが提供したテキストデータを保存する DynamoDB テーブル。
- テーブルを作成するための設定ファイル。
- Amazon Simple Notification Service トピック。
- デプロイ中にパッケージをインストールするための package.json ファイルの使用。

セクション

- 前提条件
- Elastic Beanstalk 環境の作成
- 環境内のインスタンスにアクセス許可を追加します
- サンプルアプリケーションをデプロイする
- DynamoDB テーブルを作成する
- アプリケーションの設定ファイルを更新する
- 高可用性のための環境を設定する
- クリーンアップ
- 次のステップ

前提条件

このチュートリアルでは、次の前提条件が必要です。

- Node.js ランタイム
- デフォルトの Node.js パッケージマネージャーソフトウェア、npm
- Express コマンドラインジェネレーター
- Elastic Beanstalk コマンドラインインターフェイス (EB CLI)

最初の 3 つのリストされたコンポーネントのインストールとローカル開発環境の設定の詳細については、「Elastic Beanstalk 用の Node.js 開発環境の設定」を参照してください。このチュートリアルでは、参照されるトピックでも説明されている AWS SDK for Node.js をインストールする必要はありません。

EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

Elastic Beanstalk 環境の作成

アプリケーションディレクトリ

このチュートリアルでは、アプリケーションソースバンドル用に `nodejs-example-dynamo` と呼ばれるディレクトリを使用します。このチュートリアル用の `nodejs-example-dynamo` ディレクトリを作成します。

```
~$ mkdir nodejs-example-dynamo
```

> **Note**
>
> この章の各チュートリアルでは、アプリケーションソースバンドル用に独自のディレクトリを使用します。ディレクトリ名は、チュートリアルで使用されるサンプルアプリケーションの名前と一致します。

現在の作業ディレクトリを `nodejs-example-dynamo` に変更します。

```
~$ cd nodejs-example-dynamo
```

次に、Node.js プラットフォームとサンプルアプリケーションを実行する Elastic Beanstalk 環境を設定しましょう。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。

アプリケーションの EB CLI リポジトリを設定し、Node.js プラットフォームを実行する Elastic Beanstalk 環境を作成するには

1. eb init コマンドを使用してリポジトリを作成します。

```
~/nodejs-example-dynamo$ eb init --platform node.js --region <region>
```

   このコマンドは、`.elasticbeanstalk` という名前のフォルダに、アプリケーションの環境作成用の設定ファイルを作成し、現在のフォルダに基づいた名前で Elastic Beanstalk アプリケーションを作成します。

2. eb create コマンドを使用して、サンプルアプリケーションを実行する環境を作成します。

```
~/nodejs-example-dynamo$ eb create --sample nodejs-example-dynamo
```

   このコマンドは、Node.js プラットフォームと以下のリソース用にデフォルト設定でロードバランスされた環境を作成します。

   - EC2 インスタンス – 選択したプラットフォームでウェブ・アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

     各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。

   - インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。

   - ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。

- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain.region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。
>
> Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

3. 環境の作成が完了したら、eb open コマンドを使用して、デフォルトのブラウザでその環境の URL を開きます。

```
~/nodejs-example-dynamo$ eb open
```

これで、サンプルアプリケーションを使用して Node.js Elastic Beanstalk 環境が作成されました。独自のアプリケーションで更新できます。次に、Express フレームワークを使用するようサンプルアプリケーションを更新します。

環境内のインスタンスにアクセス許可を追加します

アプリケーションは、ロードバランサーの背後で 1 つ以上の EC2 インスタンスを実行し、インターネットからの HTTP リクエストを処理します。 AWS サービスの使用を要求するリクエストを受信すると、アプリケーションは実行するインスタンスのアクセス許可を使用してそれらのサービスにアクセスします。

サンプルアプリケーションは、インスタンスのアクセス許可を使用して、データを DynamoDB テーブルに書き込み、 SDK for JavaScript in Node.js を使用して Amazon SNS トピックに通知を送信します。以下の管理ポリシーをデフォルトのインスタンスプロファイルに追加し、DynamoDB と Amazon SNS へのアクセス許可を対象環境内の EC2 インスタンスに付与します。

- AmazonDynamoDBFullAccess
- AmazonSNSFullAccess

デフォルトのインスタンスプロファイルにポリシーを追加するには

1. IAM コンソールの [ロール] ページを開きます。
2. aws-elasticbeanstalk-ec2-ロール を選択します。
3. [Permissions (アクセス許可)] タブで、[Attach policy (ポリシーの添付)] を選択します。
4. アプリケーションで使用する追加サービスの管理ポリシーを選択します。このチュートリアルでは、`AmazonSNSFullAccess` および `AmazonDynamoDBFullAccess` を選択します。
5. Attach policy] (ポリシーのアタッチ) を選択してください。

インスタンスプロファイルの詳細については、「Elastic Beanstalk インスタンスプロファイルの管理」を参照してください。

サンプルアプリケーションをデプロイする

これで、このチュートリアルのサンプルアプリケーションをデプロイして実行するために、環境の準備が整いました: nodejs-example-dynamo.zip。

チュートリアルのサンプルアプリケーションをデプロイして実行するには

1. 現在の作業ディレクトリをアプリケーションディレクトリ `nodejs-example-dynamo` に変更します。

```
~$ cd nodejs-example-dynamo
```

2. サンプルアプリケーションソースバンドル nodejs-example-dynamo.zip をアプリケーションディレクトリ `nodejs-example-dynamo` にダウンロードして内容を抽出します。
3. eb deploy コマンドを使用して、サンプルアプリケーションを Elastic Beanstalk 環境にデプロイします。

```
~/nodejs-example-dynamo$ eb deploy
```

> **Note**
>
> デフォルトでは、`eb deploy` コマンドはプロジェクトフォルダの ZIP ファイルを作成します。プロジェクトフォルダの ZIP ファイルを作成する代わりにビルドプロセスからの中間生成物をデプロイするように EB CLI を設定できます。詳細については、「プロジェクトフォルダの代わりにアーティファクトをデプロイする」を参照してください。

4. 環境の作成が完了したら、eb open コマンドを使用して、デフォルトのブラウザでその環境の URL を開きます。

```
~/nodejs-example-dynamo$ eb open
```

サイトはユーザーのお問い合わせ情報を収集し、DynamoDB テーブルを使用してデータを保存します。エントリを追加するには、[サインアップ today] を選択し、名前とメールアドレスを入力してから、[Sign Up!] を選択します。ウェブ・ アプリケーションがテーブルにフォームの内容を書き込み、Amazon SNS E メール通知をトリガーします。

現時点では、Amazon SNS トピックはプレースホルダー通知用 E メールで設定してあります。設定をまもなく更新しますが、今のところ、 AWS Management Consoleの DynamoDB テーブルおよび Amazon SNS トピックを確認できます。

テーブルを表示するには

1. DynamoDB コンソールで [テーブル] ページを開きます。
2. アプリケーションで作成したテーブルを見つけます。名前は [awseb] からスタートし、[StartupSignupsTable] を含みます。
3. テーブルを選択し、[Items] を選択してから、[Start Search] を選択してテーブルのすべての項目を表示します。

テーブルには、サインアップサイトで送信されたすべての E メールアドレスのエントリが含まれます。テーブルに対する書き込みに加えて、アプリケーションは 2 つのサブスクリプションがある Amazon SNS トピックにメッセージを送信します。1 つはお客様へのメール通知で、もう 1 つは Amazon Simple Queue Service キューです。ワーカー アプリケーションが読み取り、リクエストを処理し、関心のある顧客へ E メールを送信します。

トピックを表示するには

1. Amazon SNS コンソールで [トピック] ページを開きます。
2. アプリケーションで作成したトピックを見つけます。名前は [awseb] からスタートし、[NewSignupTopic] を含みます。
3. サブスクリプションを表示するトピックを選択します。

アプリケーション (`app.js`) には 2 つのルートの定義があります。ルートパス (/) は、名前とメールアドレスを登録するためにユーザーが入力するフォームがある Embedded JavaScript (EJS) のテンプレートでレンダリングされたウェブページを返します。フォームを送信すると、フォームデータとともに POST リクエストが `/signup` ルートへ送信され、DynamoDB テーブルにエントリが記入され、Amazon SNS トピックにサインアップの所有者を通知するメッセージをパブリッシュします。

サンプルアプリケーションには、アプリケーションが使用する DynamoDB テーブル、Amazon SNS トピック、Amazon SQS キューを作成する設定ファイルが含まれます。これにより、新しい環境を作成して機能をすぐにテストすることができますが、環境に DynamoDB テーブルを関連付ける欠点があります。実稼働環境では、環境を終了するか設定を更新するときに DynamoDB テーブルが失われないよう、環境の外部で作成する必要があります。

DynamoDB テーブルを作成する

Elastic Beanstalk で実行中のアプリケーションで外部 DynamoDB テーブルを使用するには、まず DynamoDB でテーブルを作成します。Elastic Beanstalk の外部でテーブルを作成する場合、

そのテーブルは Elastic Beanstalk と Elastic Beanstalk 環境から完全に独立しているため、Elastic Beanstalk によって終了されません。

次の設定でテーブルを作成します。

- テーブル名 – **`nodejs-tutorial`**
- プライマリ・キー – **`email`**
- プライマリ・キーのタイプ – [文字列]

DynamoDB テーブルを作成するには

1. DynamoDB マネジメントコンソールで [テーブル] ページを開きます。
2. [テーブルの作成] を選択します。
3. テーブル名 と プライマリ・キー を入力します。
4. プライマリ・キーのタイプを選択します。
5. [作成] を選択します。

アプリケーションの設定ファイルを更新する

nodejs-tutorial テーブルを使用するため、新しく作成する代わりに、アプリケーションの出典の設定ファイルを更新します。

本稼働環境用にサンプルアプリケーションを更新するには

1. 現在の作業ディレクトリをアプリケーションディレクトリ `nodejs-example-dynamo` に変更します。

   ```
   ~$ cd nodejs-example-dynamo
   ```

2. `.ebextensions/options.config` を開いて次の設定の値を変更します。

   - NewSignupEmail – お客様の E メールアドレス。
   - STARTUP_SIGNUP_TABLE – nodejs-tutorial

Example .ebextensions/options.config

```
option_settings:
  aws:elasticbeanstalk:customoption:
    NewSignupEmail: you@example.com
  aws:elasticbeanstalk:application:environment:
    THEME: "flatly"
    AWS_REGION: '`{"Ref" : "AWS::Region"}`'
    STARTUP_SIGNUP_TABLE: nodejs-tutorial
    NEW_SIGNUP_TOPIC: '`{"Ref" : "NewSignupTopic"}`'
  aws:elasticbeanstalk:container:nodejs:
    ProxyServer: nginx
  aws:elasticbeanstalk:container:nodejs:staticfiles:
    /static: /static
  aws:autoscaling:asg:
    Cooldown: "120"
  aws:autoscaling:trigger:
    Unit: "Percent"
    Period: "1"
    BreachDuration: "2"
    UpperThreshold: "75"
    LowerThreshold: "30"
    MeasureName: "CPUUtilization"
```

これにより、アプリケーションに次の設定が適用されます。

- Amazon SNS トピックが通知に使用するメールアドレスは、ユーザーのアドレス、または `options.config` ファイルに入力したアドレスに設定されます。
- `.ebextensions/create-dynamodb-table.config` によって作成されたテーブルの代わりに、nodejs-tutorial テーブルが使用されます。

3. `.ebextensions/create-dynamodb-table.config` を削除します。

```
~/nodejs-tutorial$ rm .ebextensions/create-dynamodb-table.config
```

次にアプリケーションをデプロイするとき、この設定ファイルで作成したテーブルは削除されます。

4. eb deploy コマンドを使用して、更新されたアプリケーションを Elastic Beanstalk 環境にデプロイします。

```
~/nodejs-example-dynamo$ eb deploy
```

5. 環境の作成が完了したら、eb open コマンドを使用して、デフォルトのブラウザでその環境の URL を開きます。

```
~/nodejs-example-dynamo$ eb open
```

デプロイの際、Elastic Beanstalk は Amazon SNS トピックの設定を更新し、アプリケーションの最初のバージョンをデプロイしたときに作成した DynamoDB テーブルを削除します。

これで、環境を終了するとき、[nodejs-tutorial] テーブルは削除されません。これにより、青/緑のデプロイの実行、設定ファイルの変更、またはデータ損失のリスクなしにウェブサイトの停止をすることができます。

ブラウザでサイトを開き、想定したとおりにフォームが機能することを検証します。いくつかのエントリを作成し、DynamoDB コンソールをチェックしてテーブルを検証します。

テーブルを表示するには

1. DynamoDB コンソールで [テーブル] ページを開きます。
2. [nodejs-tutorial] テーブルを探します。
3. テーブルを選択し、[Items] を選択してから、[Start Search] を選択してテーブルのすべての項目を表示します。

また、Elastic Beanstalk が以前に作成したテーブルを削除していることを確認できます。

高可用性のための環境を設定する

最後に、より高いインスタンス数で、環境の Auto Scaling グループを設定します。環境のウェブサーバーが、単一障害点となることを防ぎ、サイトをサービス停止状態にせずに変更をデプロイすることが許可されるように、常に少なくとも 2 つのインスタンスを実行します。

高可用性のために環境の Auto Scaling グループを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。

4. [容量] 設定カテゴリで、[編集] を選択します。
5. [Auto Scaling group (Auto Scaling グループ)] セクションで、[Min instances (最小インスタンス数)] を **2** に設定します。
6. ページの最下部で [適用] を選択し変更を保存します。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

また、作成した DynamoDB テーブルを削除することもできます。

DynamoDB テーブルを削除するには

1. DynamoDB コンソールで [テーブル] ページを開きます。
2. テーブルを選択します。
3. アクション を選択してから、テーブルの削除 を選択します。
4. [削除] を選択します。

次のステップ

サンプルアプリケーションは、設定ファイルを使用してソフトウェア設定を設定し、環境の一部として AWS リソースを作成します。設定ファイルとその使用方法の詳細については、「設定ファイル (`.ebextensions`) による高度な環境のカスタマイズ」を参照してください。

このチュートリアル用のサンプルアプリケーションは、Node.js のために Express ウェブフレームワークを使用しています。Express の詳細については、expressjs.com にある公式ドキュメントを参照してください。

最後に、本稼働環境でアプリケーションを使用する予定の場合は、お客様の環境設定にカスタムドメイン名を設定し、セキュアな接続のために HTTPS を有効にすることが必要になります。

Amazon RDS DB インスタンスを Node.js Elastic Beanstalk 環境に追加する

このトピックでは、Elastic Beanstalk コンソールを使用して Amazon RDS を作成する手順について説明します。Amazon Relational Database Service (Amazon RDS) DB インスタンスを使用して、アプリケーションによって収集および変更されたデータを保存することができます。データベースを環境に結合して Elastic Beanstalk で管理することも、分離したものとして作成して別のサービスで外部的に管理することもできます。これらの手順では、データベースは環境に結合され、Elastic Beanstalk によって管理されます。Amazon RDS と Elastic Beanstalk の統合の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

セクション

- 環境に DB インスタンスを追加
- ドライバのダウンロード
- データベースへの接続

環境に DB インスタンスを追加

お客様の環境に DB インスタンスを追加するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. DB エンジンを選択して、ユーザー名とパスワードを入力します。
6. ページの最下部で [適用] を選択し変更を保存します。

DB インスタンスの追加には約 10 分かかります。環境の更新が完了すると、DB インスタンスのホスト名とその他の接続情報は以下の環境プロパティを通じてアプリケーションに使用できるようになります。

プロパティ名	説明	プロパティ値
`RDS_HOSTNAME`	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
`RDS_PORT`	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。
`RDS_DB_NAME`	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
`RDS_USERNAME`	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。
`RDS_PASSWORD`	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

Elastic Beanstalk 環境と結合したデータベースインスタンスの設定の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

ドライバのダウンロード

の下にプロジェクトの `package.json` ファイル`dependencies` にデータベース・ドライバを追加します。

Example **`package.json`** – MySQL 使用の Express

```
{
  "name": "my-app",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "ejs": "latest",
    "aws-sdk": "latest",
    "express": "latest",
    "body-parser": "latest",
    "mysql": "latest"
  },
  "scripts": {
    "start": "node app.js"
  }
}
```

Node.js の共通ドライバパッケージ

- MySQL – mysql
- PostgreSQL – ノード-postgres
- SQL Server – ノード-mssql
- Oracle – ノード-oracledb

データベースへの接続

Elastic Beanstalk は、環境プロパティでアタッチされた DB インスタンスの接続情報を提供します。`process.env.`*`VARIABLE`* を使用してプロパティを読み取り、データベース接続を設定します。

Example app.js – MySQL データベースの接続

```
var mysql = require('mysql');

var connection = mysql.createConnection({
  host     : process.env.RDS_HOSTNAME,
  user     : process.env.RDS_USERNAME,
  password : process.env.RDS_PASSWORD,
  port     : process.env.RDS_PORT
```

```
});

connection.connect(function(err) {
  if (err) {
    console.error('Database connection failed: ' + err.stack);
    return;
  }

  console.log('Connected to database.');
});

connection.end();
```

ノード-mysql を使用して接続文字列を作成する方法の詳細については、npmjs.org/package/mysql を参照してください。

Node.js ツールとリソース

Node.js アプリケーションを開発するときに役に立つ参照先を次に示します。

リソース	説明
GitHub	GitHub を使用して AWS SDK for Node.js をインストールします。
AWS SDK for Node.js (開発者プレビュー)	サンプルコード、ドキュメント、ツール、追加リソースを 1 か所で入手できる場所です。

Elastic Beanstalk での PHP アプリケーションのデプロイ

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) または Elastic Beanstalk コンソールを使用して、数分で PHP アプリケーションをデプロイできます。その方法については、QuickStart チュートリアルから始めて、詳細な例を確認してください。

PHP アプリケーションを Elastic Beanstalk にデプロイするためのリソース

- PHP の QuickStart — Step-by-stepの手順。
- Elastic Beanstalk PHP プラットフォームを使用する — PHP 専用のプラットフォーム機能の使用方法。
- 高度な の例 — 一般的な PHP フレームワークとアプリケーションのチュートリアルと、Amazon RDS データベースを環境に追加する方法。

での PHP を使用した開発の詳細については AWS、次のリソースを参照してください。

- GitHub — GitHub を使用して AWS SDK for PHP をインストールします。
- PHP デベロッパーセンター — PHP アプリケーションを開発するためのツール、ドキュメント、サンプルコード AWS。
- AWS SDK for PHP FAQs — よくある質問に対する回答を取得します。

QuickStart: Elastic Beanstalk に PHP アプリケーションをデプロイする

次のチュートリアルでは、EB CLI を使用してサンプル PHP アプリケーションを作成して AWS Elastic Beanstalk 環境にデプロイする方法について説明します。

> **警告 - 本番稼働用ではありません。**
>
> 例はデモンストレーションのみを目的としています。サンプルアプリケーションを本番環境で使用しないでください。

AWS アカウント

まだ AWS のお客様でない場合は、Elastic Beanstalk を使用するための AWS アカウントを作成する必要があります。

AWS アカウントを作成する

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、通話呼び出し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するように求められます。

   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、 のセキュリティを確保し AWS IAM Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 「 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

- Elastic Beanstalk コマンドラインインターフェイス - インストールについては、「」を参照してくださいセットアップスクリプトを使用して EB CLI をインストールする (推奨) 。
- PHP - PHP ウェブサイトのインストールと設定の手順に従って、ローカルマシンに PHP をインストールします。

ステップ 1: PHP アプリケーションを作成する

このクイックスタートでは、Hello World PHP アプリケーションを作成します。

プロジェクトディレクトリを作成します。

```
~$ mkdir eb-php
~$ cd eb-php
```

次に、プロジェクトディレクトリに `index.php` ファイルを作成し、次のコードを追加します。

Example **`index.php`**

```
<?php
  echo "Hello from a PHP application running in Elastic Beanstalk!";
?>
```

ステップ 2: アプリケーションをローカルに実行する

次のコマンドを使用して、アプリケーションをローカルで実行します。

```
php -S localhost:5000
```

ブラウザを に開きますhttp://localhost:5000。

ブラウザに hello メッセージが表示され、ターミナルにメッセージがログに記録されます。

を入力してローカルサーバーを停止し`Control+c`、Elastic Beanstalk をデプロイできるようにします。

ステップ 3: PHP アプリケーションを初期化してデプロイする

次に、Elastic Beanstalk コンソールまたは EB CLI を使用してアプリケーションを環境にデプロイします。このチュートリアルでは、インタラクティブオプションで EB CLI を使用して環境を初期化します。

環境を初期化して環境を作成するには

1. 次の init コマンドを実行します。

   ```
   eb init -i
   ```

   init コマンドは、アプリケーションをインタラクティブに作成します。アプリケーション名はデフォルトでローカルフォルダ になります`eb-php`。

   SSH アクセスを除くすべてのプロンプトで、デフォルトの を受け入れて、最新の PHP プラットフォームバージョンで環境を作成します。インスタンスのトラブルシューティングでは、後で`eb init -i`コマンドを再実行するか、Amazon EC2 Instance Connect または Session Manager を使用して接続することで、SSH アクセスを設定できます。

2. 環境を作成してアプリケーションをデプロイする

   次のコマンドを実行して、 という名前の環境を作成します`blue-env`。

   ```
   eb create blue-env
   ```

   eb create コマンドを初めて実行すると、Elastic Beanstalk はソースバンドルと呼ばれるアプリケーションの zip ファイルを自動的に構築します。次に、Elastic Beanstalk は 1 つ以上の Amazon EC2 インスタンスを持つ環境を作成し、アプリケーションを環境にデプロイします。

   アプリケーションを Elastic Beanstalk にデプロイするには、最大 5 分かかる場合があります。

ステップ 4: クラウドアプリケーションを参照する

環境を作成するプロセスが完了すると、アプリケーションが実行され、ポート 5000 でリクエストをリッスンしている必要があります。次のコマンドを使用してアプリケーションに接続します。

```
eb open
```

eb open コマンドは、アプリケーション用に作成されたカスタムサブドメインにブラウザタブを開きます。

ステップ 5: アプリケーションを更新して再デプロイする

アプリケーションを作成して環境にデプロイしたら、アプリケーションの新しいバージョンまたは別のアプリケーションをいつでもデプロイできます。新しいアプリケーションバージョンのデプロイは、Amazon EC2 インスタンスのプロビジョニングや再起動を必要としないため、より高速です。

PHP コードを更新して、サーバー環境の REQUEST_TIME 値を含めます。

```
<?php
  echo "Hello from a PHP application running in Elastic Beanstalk!";

  $timestamp = $_SERVER['REQUEST_TIME'];
  echo '<br/>Request time: ' . date('Y/m/d H:i:s', $timestamp);
?>
```

次のコマンドを使用して、PHP コードを Elastic Beanstalk に再デプロイします。

```
eb deploy
```

eb deploy を実行すると、EB CLI はプロジェクトディレクトリのコンテンツをバンドルアップして、ユーザーの環境にデプロイします。

デプロイが完了したら、ページを更新するか、 を使用してアプリケーションに再接続しますeb open。更新が表示されます。そうでない場合は、ローカルサーバーを再度実行してトラブルシューティングを行い、変更を確認します。

> おめでとうございます。
>
> Elastic Beanstalk で PHP アプリケーションを作成、デプロイ、更新しました。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

または、次のコマンドを使用して EB CLI で環境を終了することもできます。

```
eb terminate
```

次のステップ

Elastic Beanstalk コンソールを使用してアプリケーション環境を探索できます。詳細については、「Explore your environment」を参照してください。

PHP を使用した詳細な例については、「」を参照してくださいElastic Beanstalk での PHP の高度な例。

Elastic Beanstalk PHP プラットフォームを使用する

AWS Elastic Beanstalk は、PHP のさまざまなバージョンのさまざまなプラットフォームブランチを提供し、サポートしています。プラットフォームは、スタンドアロンまたは Composer で実行される PHP ウェブアプリケーションをサポートします。サポートされているプラットフォームブランチの完全なリストについては、「AWS Elastic Beanstalk プラットフォーム」ドキュメントの「PHP」を参照してください。

Elastic Beanstalk には、Elastic Beanstalk 環境の Amazon EC2 インスタンスで実行されるソフトウェアをカスタマイズするために使用できる設定オプションが用意されています。アプリケーショ

ンに必要な環境変数を設定し、Amazon S3 に対してログのローテーションを有効にしたら、アプリケーションの出典で静的ファイルが含まれるフォルダを、プロキシサーバーによって提供されるパスにマッピングし、一般的な PHP 初期化設定を行うことができます。

設定オプションは実行中の環境の設定を変更するために Elastic Beanstalk コンソールで利用できます。環境を終了したときにその設定が失われないようにするため、保存済み設定を使用して設定を保存し、それを後で他の環境に適用することができます。

ソースコードの設定を保存する場合、設定ファイルを含めることができます。設定ファイルの設定は、環境を作成するたびに、またはアプリケーションをデプロイするたびに適用されます。設定ファイルを使用して、デプロイの間にパッケージをインストールしたり、スクリプトを実行したり、他のインスタンスのカスタマイズオペレーションを実行することもできます。

Composer を使用すると、デプロイの間にパッケージをインストールするために、出典バンドルに `composer.json` ファイルを含めることができます。

設定オプションとして提供されない高度な PHP 設定、および PHP の設定の場合、設定ファイルを使用して、Elastic Beanstalk により適用されるデフォルト設定を拡張および上書きできる `INI` ファイルの提供および追加拡張ができます。

Elastic Beanstalk コンソールで適用される設定は、設定ファイルに同じ設定があれば、それらの設定を上書きします。これにより、設定ファイルでデフォルト設定を定義し、コンソールでそのデフォルト設定を環境固有の設定で上書きできます。設定の優先順位の詳細と設定の他の変更方法については、「設定オプション」を参照してください。

Elastic Beanstalk Linux ベースのプラットフォームを拡張するさまざまな方法の詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

PHP プラットフォームのトピック

- AWS SDK for PHP のインストール
- PHP 8.1 on Amazon Linux 2 に関する考慮事項
- PHP 環境の設定
- 設定の名前空間
- Elastic Beanstalk PHP アプリケーションの依存関係のインストール
- Elastic Beanstalk での Composer の更新
- Elastic Beanstalk 設定での php.ini の拡張

AWS SDK for PHP のインストール

アプリケーション内から AWS リソースを管理する必要がある場合は、 をインストールします AWS SDK for PHP。例えば、SDK for PHP では、Amazon DynamoDB (DynamoDB) を使用して、リレーショナルデータベースを作成せずに、ユーザーとセッション情報を保存できます。

Composer で SDK for PHP をインストールするには

```
$ composer require aws/aws-sdk-php
```

詳細については、AWS SDK for PHP ホームページを参照してください。手順については、「 のインストール AWS SDK for PHP」を参照してください。

PHP 8.1 on Amazon Linux 2 に関する考慮事項

PHP 8.1 on Amazon Linux 2 プラットフォームブランチを使用している場合は、このセクションをお読みください。

PHP 8.1 on Amazon Linux 2 に関する考慮事項

> **Note**
>
> このトピックの情報は、PHP 8.1 on Amazon Linux 2 プラットフォームブランチにのみ適用されます。AL2023 ベースの PHP プラットフォームブランチには適用されません。また、Amazon Linux 2 ベースの PHP 8.0 プラットフォームブランチにも適用されません。

Elastic Beanstalk では、PHP 8.1 on Amazon Linux 2 プラットフォームブランチの PHP 8.1 関連の RPM パッケージは、Amazon Linux ではなくローカルディレクトリの EC2 インスタンスに保存されます。rpm -i を使用してパッケージをインストールできます。PHP 8.1 プラットフォームバージョン 3.5.0 以降、Elastic Beanstalk は PHP 8.1 関連の RPM パッケージを次のローカル EC2 ディレクトリに保存します。

```
/opt/elasticbeanstalk/RPMS
```

次の例では、php-debuginfo パッケージをインストールします。

```
$rpm -i /opt/elasticbeanstalk/RPMS/php-debuginfo-8.1.8-1.amzn2.x86_64.rpm
```

パッケージ名のバージョンは、EC2 ローカルディレクトリ `/opt/elasticbeanstalk/RPMS` にリストされている実際のバージョンによって異なります。同じ構文を使用して、他の PHP 8.1 RPM パッケージをインストールします。

次のセクションを展開すると、提供されている RPM パッケージのリストが表示されます。

RPM パッケージ

次のリストは、Elastic Beanstalk PHP 8.1 プラットフォームが Amazon Linux 2 で提供する RMP パッケージを示しています。これらのファイルは、ローカルディレクトリ `/opt/elasticbeanstalk/RPMS` にあります。

記載されているパッケージ名のバージョン番号 8.1.8-1 と 3.7.0-1 はあくまでも一例です。

- `php-8.1.8-1.amzn2.x86_64.rpm`
- `php-bcmath-8.1.8-1.amzn2.x86_64.rpm`
- `php-cli-8.1.8-1.amzn2.x86_64.rpm`
- `php-common-8.1.8-1.amzn2.x86_64.rpm`
- `php-dba-8.1.8-1.amzn2.x86_64.rpm`
- `php-dbg-8.1.8-1.amzn2.x86_64.rpm`
- `php-debuginfo-8.1.8-1.amzn2.x86_64.rpm`
- `php-devel-8.1.8-1.amzn2.x86_64.rpm`
- `php-embedded-8.1.8-1.amzn2.x86_64.rpm`
- `php-enchant-8.1.8-1.amzn2.x86_64.rpm`
- `php-fpm-8.1.8-1.amzn2.x86_64.rpm`
- `php-gd-8.1.8-1.amzn2.x86_64.rpm`
- `php-gmp-8.1.8-1.amzn2.x86_64.rpm`
- `php-intl-8.1.8-1.amzn2.x86_64.rpm`
- `php-ldap-8.1.8-1.amzn2.x86_64.rpm`
- `php-mbstring-8.1.8-1.amzn2.x86_64.rpm`
- `php-mysqlnd-8.1.8-1.amzn2.x86_64.rpm`
- `php-odbc-8.1.8-1.amzn2.x86_64.rpm`
- `php-opcache-8.1.8-1.amzn2.x86_64.rpm`

- `php-pdo-8.1.8-1.amzn2.x86_64.rpm`
- `php-pear-1.10.13-1.amzn2.noarch.rpm`
- `php-pgsql-8.1.8-1.amzn2.x86_64.rpm`
- `php-process-8.1.8-1.amzn2.x86_64.rpm`
- `php-pspell-8.1.8-1.amzn2.x86_64.rpm`
- `php-snmp-8.1.8-1.amzn2.x86_64.rpm`
- `php-soap-8.1.8-1.amzn2.x86_64.rpm`
- `php-sodium-8.1.8-1.amzn2.x86_64.rpm`
- `php-xml-8.1.8-1.amzn2.x86_64.rpm`
- `php-pecl-imagick-3.7.0-1.amzn2.x86_64.rpm`
- `php-pecl-imagick-debuginfo-3.7.0-1.amzn2.x86_64.rpm`
- `php-pecl-imagick-devel-3.7.0-1.amzn2.noarch.rpm`

PEAR パッケージと PECL パッケージを使用して、一般的な拡張機能をインストールできます。PEAR の詳細については、PEAR PHP 拡張機能とアプリケーションリポジトリのウェブサイトを参照してください。PECL の詳細については、PECL 拡張モジュールのウェブサイトを参照してください。

以下のコマンド例では、Memcached 拡張機能をインストールします。

```
$pecl install memcache
```

また、以下のコマンドを使用することもできます。

```
$pear install pecl/memcache
```

以下のコマンド例では、Redis 拡張機能をインストールします。

```
$pecl install redis
```

また、以下のコマンドを使用することもできます。

```
$pear install pecl/redis
```

PHP 環境の設定

Elastic Beanstalk コンソールでは、Amazon S3 のログローテーションを有効にしたり、アプリケーションが環境から読み取ることができる変数を設定したり、PHP 設定を変更することができます。

Elastic Beanstalk コンソールで PHP 環境を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

PHP 設定

- [プロキシサーバー] – 環境インスタンスで使用するプロキシサーバーです。デフォルトでは、nginx が使用されます。
- Document root (ドキュメントルート) – サイトのデフォルトページを含むフォルダー。ウェルカムページが、出典バンドルのルートにない場合は、ルートパスに関連して、それを含むフォルダを指定します。たとえば、ウェルカムページが、`/public` というフォルダにある場合は、`public` にします。
- Memory limit (メモリ制限) – スクリプトが割り当て許可される最大メモリ容量。例えば、`512M`。
- Zlib output compression (Zlib 出力の圧縮) – `On` に設定して、レスポンスを圧縮します。
- Allow URL fopen (URL fopen を許可) – `Off` に設定して、スクリプトがリモートの場所からファイルをダウンロードすることを防ぎます。
- Display errors (表示エラー) – `On` に設定して、デバッグの内部エラーメッセージを表示します。
- Max execution time (最大実行時間) – 環境によって終了されるまでの、スクリプトを実行許可される最大時間 (秒単位)。

ログオプション

[ログ Options] セクションには、2 つの設定があります。

- [Instance profile] – アプリケーションに関連付けられた Amazon S3 バケットへのアクセス許可が付与されているインスタンスプロファイルを指定します。

- [Enable log file rotation to Amazon S3] (Amazon S3 へのログファイルのローテーションの有効化) - アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーするかどうかを指定します。

静的ファイル

パフォーマンスを向上させるために、[Static files] (静的ファイル) セクションを使用して、ウェブアプリケーション内のディレクトリセットから静的ファイル (HTML、イメージなど) を配信するようにプロキシサーバーを設定することができます。ディレクトリごとに、仮想パスをディレクトリマッピングに設定します。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接　　処理します。

設定ファイルまたは、Elastic Beanstalk コンソールを使用した静的ファイルの設定の詳細については、「the section called “静的ファイル”」を参照してください。

環境プロパティ

環境プロパティ セクションでは、アプリケーションを実行している Amazon EC2 インスタンスの環境設定を指定できます。これらの設定は、キーバリューのペアでアプリケーションに渡されます。

アプリケーションコードは、`$_SERVER` または `get_cfg_var` 関数を使用して環境プロパティにアクセスできます。

```
$endpoint = $_SERVER['API_ENDPOINT'];
```

詳細については「環境変数およびその他のソフトウェア設定」を参照してください。

設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクをパフォーマンスできます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

次の名前空間は、プロキシサービスと PHP 固有のオプションの両方を設定します。

- `aws:elasticbeanstalk:environment:proxy:staticfiles` – 静的ファイルを提供するように環境プロキシを設定します。アプリケーションディレクトリへの仮想パスのマッピングを定義します。

- `aws:elasticbeanstalk:environment:proxy` – 環境のプロキシサーバーを指定します。
- `aws:elasticbeanstalk:container:php:phpini` – PHP 固有のオプションを設定します。この名前空間には、Elastic Beanstalk コンソールでは使用できない `composer_options` が含まれます。このオプションは、`composer.phar install` コマンドで Composer を使用して依存関係をインストールするときに使用するカスタムオプションを設定します。使用可能なオプションなど、このコマンドの詳細については、getcomposer.org ウェブサイトで「インストール」を参照してください。

次の設定ファイルの例では、`staticimages` という名前のディレクトリをパス `/images` にマップする静的ファイルオプションを指定し、`aws:elasticbeanstalk:container:php:phpini` 名前空間で使用できる各オプションの設定を示します。

Example .ebextensions/php-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /images: staticimages
  aws:elasticbeanstalk:container:php:phpini:
    document_root: /public
    memory_limit: 128M
    zlib.output_compression: "Off"
    allow_url_fopen: "On"
    display_errors: "Off"
    max_execution_time: 60
    composer_options: vendor/package
```

> **Note**
>
> `aws:elasticbeanstalk:environment:proxy:staticfiles` 名前空間は、Amazon Linux AMI PHP プラットフォームブランチ (Amazon Linux 2 以前) では定義されていません。

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

Elastic Beanstalk PHP アプリケーションの依存関係のインストール

このトピックでは、必要な他の PHP パッケージをインストールするようにアプリケーションを設定する方法について説明します。アプリケーションは、他の PHP パッケージに依存関係がある可能性があります。環境の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスにこれらの依存関係をインストールするようにアプリケーションを設定することができます。または、アプリケーションの依存関係を出典バンドルに含め、アプリケーションを用いてデプロイすることができます。このセクションでは、これら両方の方法について説明します。

Composer ファイルを使用してインスタンスに依存関係をインストールする

プロジェクト出典のルートで `composer.json` ファイルを使用し、環境の Amazon EC2 インスタンスでアプリケーションに必要なパッケージを composer を使用してインストールします。

Example composer.json

```
{
    "require": {
        "monolog/monolog": "1.0.*"
    }
}
```

`composer.json` ファイルがある場合、Elastic Beanstalk は `composer.phar install` を実行して依存関係をインストールします。`aws:elasticbeanstalk:container:php:phpini` 名前空間で `composer_options` のオプションを設定することにより、オプションを追加して、コマンドに追加できます。

出典バンドルに依存関係を含める

アプリケーションに多数の依存関係がある場合、インストールに長い時間がかかる場合があります。依存関係は新しいインスタンスにインストールされるため、これによりデプロイおよびスケーリング・オペレーションが増える可能性があります。

デプロイ時間に悪影響が発生しないようにするには、開発環境で Composer を使用して依存関係を解決し、`vendor` フォルダにインストールします。

アプリケーション出典バンドルに依存関係を含めるには

1. 次のコマンドを実行してください。

```
% composer install
```

2. 生成された vendor フォルダをアプリケーション出典バンドルのルートに含めます。

Elastic Beanstalk がインスタンスで vendor フォルダを検出した場合、composer.json ファイルは (存在する場合でも) 無視されます。アプリケーションは vendor フォルダから依存関係を使用します。

Elastic Beanstalk での Composer の更新

このトピックでは、Composer を最新の状態に保つように Elastic Beanstalk を設定する方法について説明します。Composer ファイルを含むパッケージをインストールする際にエラーが表示される場合、または最新のプラットフォームバージョンを使用できない場合は、Composer を更新する必要がある場合があります。プラットフォーム更新の間に、.ebextensions フォルダの設定ファイルを使用して環境のインスタンスにある Composer を更新できます。

Composer は、次の設定で自己更新できます。

```
commands:
  01updateComposer:
    command: /usr/bin/composer.phar self-update 2.7.0
```

次のオプション設定では、Composer キャッシュの場所を設定する COMPOSER_HOME 環境変数が設定されます。

```
option_settings:
  - namespace: aws:elasticbeanstalk:application:environment
    option_name: COMPOSER_HOME
    value: /home/webapp/composer-home
```

これらの両方を、.ebextensions フォルダ内の同じ設定ファイルに組み合わせることができます。

Example .ebextensions/composer.config

```
commands:
  01updateComposer:
    command: /usr/bin/composer.phar self-update 2.7.0
```

```
option_settings:
  - namespace: aws:elasticbeanstalk:application:environment
    option_name: COMPOSER_HOME
    value: /home/webapp/composer-home
```

> **Note**
>
> 2024 年 2 月 22 日の AL2023 プラットフォームリリースと 2024 年 2 月 28 日の AL2 プラットフォームリリースで Composer のインストールが更新されているため、自己更新の実行時に `COMPOSER_HOME` が設定されている場合、Composer の自己更新が失敗する可能性があります。
> 次の組み合わせコマンドは実行に失敗します。`export COMPOSER_HOME=/home/webapp/composer-home && /usr/bin/composer.phar self-update 2.7.0`
> ただし、前の例は機能します。前の例では、`COMPOSER_HOME` のオプション設定は `01updateComposer` の実行に渡されず、自己更新コマンドの実行時に設定されません。

> **Important**
>
> `composer.phar self-update` コマンドでバージョン番号を省略すると、Composer は出典コードをデプロイする際、および Auto Scaling によって新しいインスタンスがプロビジョニングされる際に毎回、利用可能な最新バージョンに更新されます。そのため、アプリケーションに対応しない Composer のバージョンがリリースされていると、スケーリング・オペレーションおよびデプロイが失敗する場合があります。

Composer のバージョンを含む Elastic Beanstalk PHP Platforms の詳細については、AWS Elastic Beanstalk プラットフォームドキュメントの「PHP プラットフォームバージョン」を参照してください。

Elastic Beanstalk 設定での php.ini の拡張

`files` ブロックで設定ファイルを使用して、`.ini` ファイルを環境のインスタンスの `/etc/php.d/` に追加します。主要な設定ファイルの `php.ini` は、このフォルダにファイルの設定をアルファベット順に取得します。多くの拡張機能はこのフォルダのファイルによりデフォルトで有効になります。

Example .ebextensions/mongo.config

```
files:
```

```
  "/etc/php.d/99mongo.ini":
    mode: "000755"
    owner: root
    group: root
    content: |
      extension=mongo.so
```

Elastic Beanstalk での PHP の高度な例

で PHP アプリケーションの使用を開始するには AWS Elastic Beanstalk、 環境にデプロイする最初のアプリケーションバージョンとしてアップロードするアプリケーションソースバンドルが必要です。

EB CLI でデプロイされたシンプルな PHP アプリケーションの使用を開始するPHP の QuickStartには、 をお勧めします。

高度な PHP の例

- PHP Elastic Beanstalk 環境に Amazon RDS DB インスタンスを追加する
- Elastic Beanstalk への Laravel アプリケーションのデプロイ
- Elastic Beanstalk への CakePHP アプリケーションのデプロイ
- Elastic Beanstalk への Symfony アプリケーションのデプロイ
- 外部 Amazon RDS データベースを使用して高可用性の PHP アプリケーションを Elastic Beanstalk にデプロイする
- 外部 Amazon RDS データベースを使用して高可用性の WordPress ウェブサイトを Elastic Beanstalk にデプロイする
- 外部 Amazon RDS データベースを使用して高可用性の Drupal ウェブサイトを Elastic Beanstalk にデプロイする

PHP Elastic Beanstalk 環境に Amazon RDS DB インスタンスを追加する

このトピックでは、Elastic Beanstalk コンソールを使用して Amazon RDS を作成する手順について説明します。Amazon Relational Database Service (Amazon RDS) DB インスタンスを使用して、アプリケーションによって収集および変更されたデータを保存することができます。データベースを環境に結合して Elastic Beanstalk で管理することも、分離したものとして作成して別のサービスで外部的に管理することもできます。これらの手順では、データベースは環境に結合され、Elastic

Beanstalk によって管理されます。Amazon RDS と Elastic Beanstalk の統合の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

セクション

- 環境に DB インスタンスを追加
- ドライバのダウンロード
- PDO または MySQLi を使用してデータベースに接続
- Symfony を使用してデータベースに接続する

環境に DB インスタンスを追加

お客様の環境に DB インスタンスを追加するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. DB エンジンを選択して、ユーザー名とパスワードを入力します。
6. ページの最下部で [適用] を選択し変更を保存します。

DB インスタンスの追加には約 10 分かかります。環境の更新が完了すると、DB インスタンスのホスト名とその他の接続情報は以下の環境プロパティを通じてアプリケーションに使用できるようになります。

プロパティ名	説明	プロパティ値
`RDS_HOSTNAME`	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
`RDS_PORT`	DB インスタンスが接続を許可するポート。デフォルト値は	Amazon RDS コンソールの [Connectivity & security (接続

プロパティ名	説明	プロパティ値
	DB エンジンによって異なります。	とセキュリティ)] タブ: [Port (ポート)]。
RDS_DB_NAME	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
RDS_USERNAME	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。
RDS_PASSWORD	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

Elastic Beanstalk 環境と結合したデータベースインスタンスの設定の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

ドライバのダウンロード

PHP データオブジェクト (PDO) を使用してデータベースに connect するには、選択したデータベースエンジンに一致するドライバーをインストールします。

- MySQL – PDO_MYSQL
- PostgreSQL – PDO_PGSQL
- Oracle – PDO_OCI
- SQL Server – PDO_SQLSRV

詳細については、「http://php.net/manual/en/pdo.installation.php」を参照してください。

PDO または MySQLi を使用してデータベースに接続

$_SERVER[`*VARIABLE*`] を使用して、環境から接続情報の読み取りができます。

PDO では、ホスト、ポート、名前からデータソース名 (DSN) を作成します。データベースのユーザー名とパスワードを使用して、PDO のコンストラクタに DSN を渡します。

Example PDO - MySQL を使用して RDS データベースに connect する

```
<?php
$dbhost = $_SERVER['RDS_HOSTNAME'];
$dbport = $_SERVER['RDS_PORT'];
$dbname = $_SERVER['RDS_DB_NAME'];
$charset = 'utf8' ;

$dsn = "mysql:host={$dbhost};port={$dbport};dbname={$dbname};charset={$charset}";
$username = $_SERVER['RDS_USERNAME'];
$password = $_SERVER['RDS_PASSWORD'];

$pdo = new PDO($dsn, $username, $password);
?>
```

他のドライバーについては、`mysql` をドライバー (`pgsql`、`oci`、または `sqlsrv`) の名前に置き換えます。

MySQLi については、ホスト名、ユーザー名、パスワード、データベース名、およびポートを `mysqli` コンストラクタに渡します。

Example mysqli_connect() を使用して RDS データベースに接続する

```
$link = new mysqli($_SERVER['RDS_HOSTNAME'], $_SERVER['RDS_USERNAME'],
 $_SERVER['RDS_PASSWORD'], $_SERVER['RDS_DB_NAME'], $_SERVER['RDS_PORT']);
```

Symfony を使用してデータベースに接続する

Symfony バージョン 3.2 以降については、`%env(PROPERTY_NAME)%` を使用し、Elastic Beanstalk で設定された環境プロパティに基づいて設定ファイルのデータベースパラメータを設定します。

Example app/config/parameters.yml

```
parameters:
    database_driver:   pdo_mysql
    database_host:     '%env(RDS_HOSTNAME)%'
    database_port:     '%env(RDS_PORT)%'
    database_name:     '%env(RDS_DB_NAME)%'
    database_user:     '%env(RDS_USERNAME)%'
    database_password: '%env(RDS_PASSWORD)%'
```

詳細については、外部パラメータ (Symfony 3.4) に関する記事を参照してください。

旧バージョンの Symfony については、SYMFONY__ で始まる環境変数にのみアクセスできます。Elastic Beanstalk 定義の環境プロパティにはアクセスできません。また、Symfony に接続情報を渡すには、独自の環境プロパティを定義する必要があります。

Symfony 2 でデータベースに connect するには、パラメータごとに環境プロパティを作成します。次に、%*property.name*% を使用して、設定ファイルの Symfony で変換された可変数にアクセスします。たとえば、SYMFONY__DATABASE__USER という名前の環境プロパティには database.user としてアクセスできます。

```
database_user:     "%database.user%"
```

詳細については、外部パラメータ (Symfony 2.8) に関する記事を参照してください。

Elastic Beanstalk への Laravel アプリケーションのデプロイ

Laravel は PHP 用のオープンソースのモデルビュー・コントローラー（MVC）フレームワークです。このチュートリアルでは、Laravel アプリケーションを生成して AWS Elastic Beanstalk 環境にデプロイし、Amazon Relational Database Service (Amazon RDS) データベースインスタンスに接続するように設定するプロセスについて説明します。

セクション

- 前提条件
- Elastic Beanstalk 環境の起動
- Laravel をインストールしてウェブサイトを生成する
- アプリケーションをデプロイします
- Composer 設定の設定
- お客様の環境にデータベースを追加する
- クリーンアップ
- 次のステップ

前提条件

このチュートリアルでは、基本的な Elastic Beanstalk オペレーションと Elastic Beanstalk コンソールに関する知識があることを前提としています。まだ起動していない場合は、Elastic Beanstalk の使用を開始する方法について説明します。 の指示に従って、最初の Elastic Beanstalk 環境を起動します。

このガイドの手順に従うには、run command のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

Laravel 6 では PHP 7.2 以降が必要です。また、Laravel 公式ドキュメントのサーバー要件のトピックに記載されている PHP 拡張も必要です。手順に従って PHP と Composer をインストールします。

Laravel のサポートおよびメンテナンス情報については、Laravel 公式ドキュメントの support ポリシーのトピックをご参照ください。

Elastic Beanstalk 環境の起動

Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境を作成します。[PHP] プラットフォームを選択し、デフォルト設定およびサンプルコードを受け入れます。

環境を起動するには (コンソール)

1. 事前に設定されたリンク (console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced) を使用して、Elastic Beanstalk コンソールを開きます。
2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。
3. アプリケーションコード として、サンプルアプリケーション を選択します。
4. 確認と起動 を選択します。
5. 使用できるオプションを確認します。使用する有効なオプションを選択し、準備ができたら [アプリケーションの作成] を選択します。

環境の作成の所要時間は約 5 分です。以下のリソースが作成されます。

- EC2 インスタンス – 選択したプラットフォームでウェブ・アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。
> Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

これらのリソースはすべて Elastic Beanstalk によって管理されます。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

> **Note**
>
> Elastic Beanstalk が作成する Amazon S3 バケットは、環境間で共有され、環境の終了時に削除されません。詳細については、「Amazon S3 で Elastic Beanstalk を使用する」を参照してください。

Laravel をインストールしてウェブサイトを生成する

以下の 1 つのコマンドで、Composer は Laravel をインストールして作業プロジェクトを作成できます。

```
~$ composer create-project --prefer-dist laravel/laravel eb-laravel
```

Composer は Laravel とその依存関係をインストールし、デフォルトのプロジェクトを生成します。

Laravel のインストール中に問題が発生した場合は、公式ドキュメント (https://laravel.com/docs/6.x) のインストールに関するトピックをご参照ください。

アプリケーションをデプロイします

Composer で作成されたファイルを含む出典バンドルを作成します。次のコマンドでは、`laravel-default.zip` という出典バンドルが作成されます。`vendor` フォルダ内のファイルは除外され

ます。これらのファイルは、多くのスペースを使用するだけでなく、アプリケーションを Elastic Beanstalk にデプロイするのに不要です。

```
~/eb-laravel$ zip ../laravel-default.zip -r * .[^.]* -x "vendor/*"
```

ソースバンドルを Elastic Beanstalk にアップロードし、Laravel を環境にデプロイします。

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

> **Note**
>
> さらにソースバンドルを最適化するには、Git リポジトリを初期化し、`git archive` コマンドを使用して出典バンドルを作成します。デフォルトの Laravel プロジェクトには、`.gitignore` ファイルが含まれています。このファイルは、デプロイに不要な `vendor` フォルダと他のファイルを除外するよう Git に指示します。

Composer 設定の設定

デプロイが完了したら、URL をクリックして Laravel アプリケーションをブラウザで開きます。

説明 デフォルトでは、Elastic Beanstalk によってウェブサイトのルートパスにプロジェクトのルートが提供されています。この場合、デフォルトのページ (`index.php`) は `public` フォルダーの 1 レベル下にあります。URL に `/public` を追加することでこれを確認できます。例えば、`http://laravel.us-east-2.elasticbeanstalk.com/public`。

ルートパスで Laravel アプリケーションをサポートするには、Elastic Beanstalk コンソールを使用してウェブサイトのドキュメントルートを設定します。

ウェブサイトのドキュメントルートを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [ドキュメントのルート] に「**/public**」と入力します。
6. ページの最下部で [適用] を選択し変更を保存します。
7. 更新が完了したら、ブラウザーでサイトを再び開くための URL をクリックします。

ここまで完了したら、次は、お客様の環境にデータベースを追加し、そのデータベースに接続するように Laravel を設定します。

お客様の環境にデータベースを追加する

Elastic Beanstalk 環境で RDS DB インスタンスを起動します。Elastic Beanstalk 上の Laravel では、MySQL、SQLServer、または PostgreSQL データベースを使用できます。この例では、MySQL を使用します。

RDS DB インスタンスを Elastic Beanstalk 環境に追加するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. [エンジン] で、[mysql] を選択します。
6. マスターの [username] と [password] に入力します。Elastic Beanstalk は環境プロパティを使用して、アプリケーションにこれらの値を渡します。
7. ページの最下部で [適用] を選択し変更を保存します。

データベースインスタンスの作成には約 10 分かかります。Elastic Beanstalk 環境に結合されたデータベースの詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

その間に、環境から接続情報を読み取るようにソースコードを更新できます。Elastic Beanstalk は、RDS_HOSTNAME などの環境変数を使用して、接続の詳細をアプリケーションに渡します。

Laravel のデータベース設定は、プロジェクトコードで database.php フォルダ内の config という名前のファイルに保存されています。mysql エントリを見つけ、host、database、username、 and password の各変数を変更して Elastic Beanstalk から対応する値を読み取ります。

Example ~/Eb-laravel/config/database.php

```
...
    'connections' => [

        'sqlite' => [
            'driver' => 'sqlite',
            'database' => env('DB_DATABASE', database_path('database.sqlite')),
            'prefix' => '',
        ],

        'mysql' => [
            'driver' => 'mysql',
            'host' => env('RDS_HOSTNAME', '127.0.0.1'),
            'port' => env('RDS_PORT', '3306'),
            'database' => env('RDS_DB_NAME', 'forge'),
            'username' => env('RDS_USERNAME', 'forge'),
            'password' => env('RDS_PASSWORD', ''),
            'unix_socket' => env('DB_SOCKET', ''),
            'charset' => 'utf8mb4',
            'collation' => 'utf8mb4_unicode_ci',
            'prefix' => '',
            'strict' => true,
            'engine' => null,
        ],
...
```

データベース接続が正しく設定されていることを確認するには、データベースに connect してデフォルトのレスポンスを返すコードを index.php に追加します。

Example ~/Eb-laravel/公開/index.php

```
...
if(DB::connection()->getDatabaseName())
```

```
{
   echo "Connected to database ".DB::connection()->getDatabaseName();
}
$response->send();
...
```

DB インスタンスの起動が完了したら、更新したアプリケーションのバンドルを作成し、お客様の環境にデプロイします。

Elastic Beanstalk 環境を更新するには

1. 新しい出典バンドルを作成します。

   ```
   ~/eb-laravel$ zip ../laravel-v2-rds.zip -r * .[^.]* -x "vendor/*"
   ```

2. Elastic Beanstalk 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。
3. ナビゲーションペインで、[環境] を選択し、リストから環境の名前を選択します。
4. [アップロードとデプロイ] を選択します。
5. [Browse] を選択して `laravel-v2-rds.zip` をアップロードします。
6. [Deploy] (デプロイ) を選択します。

アプリケーションの新しいバージョンのデプロイには 1 分以上かかりません。デプロイが完了したら、ウェブ・ページを更新して、データベースに接続されたことを確認します。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。

3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

さらに、Elastic Beanstalk 環境の外に作成したデータベースリソースを終了できます。Amazon RDS DB インスタンスを終了する場合、スナップショットを作成し、後で別のインスタンスにデータを復元できます。

RDS DB インスタンスを終了するには

1. Amazon RDS コンソールを開きます。
2. [データベース] を選択します。
3. DB インスタンスを選択します。
4. [アクション] を選択し、[削除] を選択します。
5. スナップショットを作成するかどうかを選択してから、削除 を選択します。

次のステップ

Laravel の詳細については、Laravel 公式ウェブサイト (laravel.com) をご覧ください。

アプリケーションの開発が進むにつれ、.zip ファイルを手動で作成して Elastic Beanstalk コンソールにアップロードすることなく、環境を管理してアプリケーションをデプロイする方法が必要になります。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) には、コマンドラインインターフェイスからアプリケーションを作成、設定して、Elastic Beanstalk 環境にデプロイするための使いやすいコマンドが用意されています。

このチュートリアルでは、Elastic Beanstalk コンソールを使用して Composer のオプションを設定しました。この設定をアプリケーション出典のパートにするには、次のような設定ファイルを使用できます。

Example .ebextensions/composer.config

```
option_settings:
  aws:elasticbeanstalk:container:php:phpini:
    document_root: /public
```

詳細については、「設定ファイル (.ebextensions) による高度な環境のカスタマイズ」を参照してください。

Elastic Beanstalk 環境で Amazon RDS DB インスタンスを実行するのは、開発とテストには最適ですが、データベースのライフサイクルがお客様の環境に結び付けられます。自身の環境の外で実行されているデータベースに接続する手順については、「PHP Elastic Beanstalk 環境に Amazon RDS DB インスタンスを追加する」を参照してください。

最後に、本稼働環境でアプリケーションを使用する予定の場合は、お客様の環境にカスタムドメイン名を設定し、安全な接続のために HTTPS を有効にすることが必要になります。

Elastic Beanstalk への CakePHP アプリケーションのデプロイ

CakePHP は PHP 用のオープンソースの MVC フレームワークです。このチュートリアルでは、CakePHP のプロジェクトを生成し、Elastic Beanstalk 環境にデプロイし、Amazon RDS データベースインスタンスに接続するように設定するプロセスについて説明します。

セクション

- 前提条件
- Elastic Beanstalk 環境の起動
- CakePHP をインストールしてウェブサイトを生成する
- アプリケーションをデプロイします
- お客様の環境にデータベースを追加する
- クリーンアップ
- 次のステップ

前提条件

このチュートリアルでは、基本的な Elastic Beanstalk オペレーションと Elastic Beanstalk コンソールに関する知識があることを前提としています。まだ起動していない場合は、Elastic Beanstalk の使用を開始する方法について説明します。 の指示に従って、最初の Elastic Beanstalk 環境を起動します。

このガイドの手順に従うには、run command のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

CakePHP 4 には PHP 7.4 以降が必要です。また、CakePHP インストールに関する公式ドキュメントに記載されている PHP 拡張も必要です。PHP と Composer の両方をインストールする必要があります。

Elastic Beanstalk 環境の起動

Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境を作成します。[PHP] プラットフォームを選択し、デフォルト設定およびサンプルコードを受け入れます。

環境を起動するには (コンソール)

1. 事前に設定されたリンク (console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced) を使用して、Elastic Beanstalk コンソールを開きます。
2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。
3. アプリケーションコード として、サンプルアプリケーション を選択します。
4. 確認と起動 を選択します。
5. 使用できるオプションを確認します。使用する有効なオプションを選択し、準備ができたら [アプリケーションの作成] を選択します。

環境の作成の所要時間は約 5 分です。以下のリソースが作成されます。

- EC2 インスタンス – 選択したプラットフォームでウェブ・ アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの

HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。

- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。
> Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

これらのリソースはすべて Elastic Beanstalk によって管理されます。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

> **Note**
>
> Elastic Beanstalk が作成する Amazon S3 バケットは、環境間で共有され、環境の終了時に削除されません。詳細については、「Amazon S3 で Elastic Beanstalk を使用する」を参照してください。

CakePHP をインストールしてウェブサイトを生成する

以下の 1 つのコマンドで、Composer は CakePHP をインストールして作業プロジェクトを作成できます。

```
~$ composer create-project --prefer-dist cakephp/app eb-cake
```

Composer は、CakePHP と約 20 個の依存関係をインストールし、デフォルトのプロジェクトを生成します。

CakePHP のインストール中に問題が発生した場合は、公式ドキュメントのインストールに関するトピック (http://book.cakephp.org/4.0/en/installation.html) をご参照ください。

アプリケーションをデプロイします

Composer で作成されたファイルを含む出典バンドルを作成します。次のコマンドでは、`cake-default.zip` という出典バンドルが作成されます。`vendor` フォルダ内のファイルは除外されます。これらのファイルは、多くのスペースを使用するだけでなく、アプリケーションを Elastic Beanstalk にデプロイするのに不要です。

```
eb-cake zip ../cake-default.zip -r * .[^.]* -x "vendor/*"
```

ソースバンドルを Elastic Beanstalk にアップロードし、CakePHP を環境にデプロイします。

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。

3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

> **Note**
>
> さらに出典バンドルを最適化するには、Git リポジトリを初期化し、`git archive` コマンドを使用して出典バンドルを作成します。デフォルトの Symfony プロジェクトには、`.gitignore` ファイルが含まれています。このファイルは、デプロイに不要な `vendor` フォルダと他のファイルを除外するよう Git に指示します。

プロセスが完了したら、ブラウザで CakePHP アプリケーションを開くための URL をクリックします。

ここまで完了したら、次は、お客様の環境にデータベースを追加し、そのデータベースに接続するように CakePHP を設定します。

お客様の環境にデータベースを追加する

Elastic Beanstalk 環境で Amazon RDS データベースインスタンスを起動します。Elastic Beanstalk 上の CakePHP では、MySQL、SQLServer、または PostgreSQL データベースを使用できます。この例では、PostgreSQL を使用します。

Amazon RDS DB インスタンスを Elastic Beanstalk 環境に追加するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [Database (データベース)] で、[Edit (編集)] を選択します。
5. [DB engine] で、[postgres] を選択します。
6. マスターの [username] と [password] に入力します。Elastic Beanstalk は環境プロパティを使用して、アプリケーションにこれらの値を渡します。
7. ページの最下部で [適用] を選択し変更を保存します。

データベースインスタンスの作成には約 10 分かかります。その間に、環境から接続情報を読み取るようにソースコードを更新できます。Elastic Beanstalk は、RDS_HOSTNAME などの環境変数を使用して、接続の詳細をアプリケーションに渡します。

CakePHP のデータベース設定は、プロジェクトコードで app.php フォルダ内の config という名前のファイルにあります。このファイルを開き、$_SERVER から環境変数を読み取ってローカル変数に割り当てるコードをいくつか追加します。最初の行 (<?php) の後に、以下の例で強調表示された行を挿入します。

Example ~/Eb-cake/config/app.php

```
<?php
if (!defined('RDS_HOSTNAME')) {
  define('RDS_HOSTNAME', $_SERVER['RDS_HOSTNAME']);
  define('RDS_USERNAME', $_SERVER['RDS_USERNAME']);
  define('RDS_PASSWORD', $_SERVER['RDS_PASSWORD']);
  define('RDS_DB_NAME', $_SERVER['RDS_DB_NAME']);
}
return [
...
```

データベース接続は app.php 内をさらに下がったセクションで設定されています。以下のセクションを見つけ、データベースエンジン (Mysql、Sqlserver、または Postgres) と一致するドライバの名前で、デフォルトのデータソース設定を変更し、Elastic Beanstalk から対応する値を読み取るように host、username、password、database 変数を設定します。

Example ~/Eb-cake/config/app.php

```
...
     /**
     * Connection information used by the ORM to connect
     * to your application's datastores.
     * Drivers include Mysql Postgres Sqlite Sqlserver
     * See vendor\cakephp\cakephp\src\Database\Driver for complete list
     */
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Postgres',
            'persistent' => false,
            'host' => RDS_HOSTNAME,
```

```
            /*
             * CakePHP will use the default DB port based on the driver selected
             * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
             * the following line and set the port accordingly
             */
            //'port' => 'non_standard_port_number',
            'username' => RDS_USERNAME,
            'password' => RDS_PASSWORD,
            'database' => RDS_DB_NAME,
            /*
             * You do not need to set this flag to use full utf-8 encoding (internal
 default since CakePHP 3.6).
             */
            //'encoding' => 'utf8mb4',
            'timezone' => 'UTC',
            'flags' => [],
            'cacheMetadata' => true,
            'log' => false,
...
```

DB インスタンスの起動が完了したら、更新したアプリケーションのバンドルを作成し、お客様の環境にデプロイします。

Elastic Beanstalk 環境を更新するには

1. 新しい出典バンドルを作成します。

   ```
   ~/eb-cake$ zip ../cake-v2-rds.zip -r * .[^.]* -x "vendor/*"
   ```

2. Elastic Beanstalk 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。
3. ナビゲーションペインで、[環境] を選択し、リストから環境の名前を選択します。
4. [アップロードとデプロイ] を選択します。
5. [Browse] を選択して `cake-v2-rds.zip` をアップロードします。
6. [Deploy] (デプロイ) を選択します。

アプリケーションの新しいバージョンのデプロイには 1 分以上かかりません。デプロイが完了したら、ウェブ・ページを更新して、データベースに接続されたことを確認します。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

さらに、Elastic Beanstalk 環境の外に作成したデータベースリソースを終了できます。Amazon RDS DB インスタンスを終了する場合、スナップショットを作成し、後で別のインスタンスにデータを復元できます。

RDS DB インスタンスを終了するには

1. Amazon RDS コンソールを開きます。
2. [データベース] を選択します。
3. DB インスタンスを選択します。
4. [アクション] を選択し、[削除] を選択します。
5. スナップショットを作成するかどうかを選択してから、削除 を選択します。

次のステップ

CakePHP の詳細については、ドキュメント (book.cakephp.org) をお読みください。

アプリケーションの開発が進むにつれ、.zip ファイルを手動で作成して Elastic Beanstalk コンソールにアップロードすることなく、環境を管理してアプリケーションをデプロイする方法が必要になります。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) には、コマンドラインインターフェイスからアプリケーションを作成、設定して、Elastic Beanstalk 環境にデプロイするための使いやすいコマンドが用意されています。

Elastic Beanstalk 環境で Amazon RDS DB インスタンスを実行するのは、開発とテストには最適ですが、データベースのライフサイクルがお客様の環境に結び付けられます。自身の環境の外で実行されているデータベースに接続する手順については、「PHP Elastic Beanstalk 環境に Amazon RDS DB インスタンスを追加する」を参照してください。

最後に、本稼働環境でアプリケーションを使用する予定の場合は、お客様の環境にカスタムドメイン名を設定し、安全な接続のために HTTPS を有効にすることが必要になります。

Elastic Beanstalk への Symfony アプリケーションのデプロイ

Symfony は、動的な PHP ウェブアプリケーションを作成するためのオープンソースフレームワークです。このチュートリアルでは、Symfony アプリケーションを生成して AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

セクション

- 前提条件
- Elastic Beanstalk 環境の起動
- Symfony をインストールしてウェブサイトを生成する
- アプリケーションをデプロイします
- Composer 設定の設定
- クリーンアップ
- 次のステップ

前提条件

このチュートリアルでは、基本的な Elastic Beanstalk オペレーションと Elastic Beanstalk コンソールに関する知識があることを前提としています。まだ起動していない場合は、Elastic Beanstalk の使用を開始する方法について説明します。 の指示に従って、最初の Elastic Beanstalk 環境を起動します。

このガイドの手順に従うには、run command のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

Symfony 4.4.9 では PHP 7.1.3 以降が必要です。また、Symfony のインストールに関する公式ドキュメントの技術的要件のトピックに記載されている PHP 拡張も必要です。このチュートリアルでは、PHP 7.2 と対応する Elastic Beanstalk プラットフォームバージョンを使用します。続行する前に、PHP と Composer の両方をインストールする必要があります。

Symfony のサポートおよびメンテナンス情報については、Symfony ウェブサイトの Symfony のリリースのトピックをご参照ください。Symfony 4.4.9 の PHP バージョンサポートに関連する更新の詳細については、Symfony ウェブサイトの Symfony 4.4.9 リリースノートのトピックをご参照ください。

Elastic Beanstalk 環境の起動

Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境を作成します。[PHP] プラットフォームを選択し、デフォルト設定およびサンプルコードを受け入れます。

環境を起動するには (コンソール)

1. 事前に設定されたリンク (console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced) を使用して、Elastic Beanstalk コンソールを開きます。
2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。
3. アプリケーションコード として、サンプルアプリケーション を選択します。
4. 確認と起動 を選択します。
5. 使用できるオプションを確認します。使用する有効なオプションを選択し、準備ができたら [アプリケーションの作成] を選択します。

環境の作成の所要時間は約 5 分です。以下のリソースが作成されます。

- EC2 インスタンス – 選択したプラットフォームでウェブ・アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを

実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。

- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

これらのリソースはすべて Elastic Beanstalk によって管理されます。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

> **Note**
>
> Elastic Beanstalk が作成する Amazon S3 バケットは、環境間で共有され、環境の終了時に削除されません。詳細については、「Amazon S3 で Elastic Beanstalk を使用する」を参照してください。

Symfony をインストールしてウェブサイトを生成する

Composer は、次のコマンド 1 つで Symfony をインストールし、作業プロジェクトを作成できます。

```
~$ composer create-project symfony/website-skeleton eb-symfony
```

Composer は Symfony とその依存関係をインストールし、デフォルトのプロジェクトを生成します。

Symfony のインストール中に問題が発生した場合は、Symfony の公式ドキュメントのインストールに関するトピックをご参照ください。

アプリケーションをデプロイします

プロジェクトディレクトリに移動します。

```
~$ cd eb-symfony
```

Composer で作成されたファイルを含む出典バンドルを作成します。次のコマンドでは、`symfony-default.zip` という出典バンドルが作成されます。`vendor` フォルダ内のファイルは除外されます。これらのファイルは、多くのスペースを使用するだけでなく、アプリケーションを Elastic Beanstalk にデプロイするのに不要です。

```
eb-symfony$ zip ../symfony-default.zip -r * .[^.]* -x "vendor/*"
```

ソースバンドルを Elastic Beanstalk にアップロードし、Symfony を環境にデプロイします。

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

> **Note**
>
> さらに出典バンドルを最適化するには、Git リポジトリを初期化し、`git archive` コマンドを使用して出典バンドルを作成します。デフォルトの Symfony プロジェクトには、`.gitignore` ファイルが含まれています。このファイルは、デプロイに不要な `vendor` フォルダと他のファイルを除外するよう Git に指示します。

Composer 設定の設定

デプロイが完了したら、URL をクリックして Symfony アプリケーションをブラウザで開きます。

説明 デフォルトでは、Elastic Beanstalk によってウェブサイトのルートパスにプロジェクトのルートが提供されています。この場合、デフォルトのページ (`app.php`) は `web` フォルダーの 1 レベル下にあります。URL に `/public` を追加することでこれを確認できます。例えば、`http://symfony.us-east-2.elasticbeanstalk.com/public`。

ルートパスで Symfony アプリケーションをサポートするには、Elastic Beanstalk コンソールを使用してウェブサイトのドキュメントルートを設定します。

ウェブサイトのドキュメントルートを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

5. [ドキュメントのルート] に「**/public**」と入力します。
6. ページの最下部で [適用] を選択し変更を保存します。
7. 更新が完了したら、ブラウザーでサイトを再び開くための URL をクリックします。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

次のステップ

Symfony の詳細については、Symfony とは何かに関する記事 (symfony.com) を参照してください。

アプリケーションの開発が進むにつれ、.zip ファイルを手動で作成して Elastic Beanstalk コンソールにアップロードすることなく、環境を管理してアプリケーションをデプロイする方法が必要になります。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) には、コマンドラインインターフェイスからアプリケーションを作成、設定して、Elastic Beanstalk 環境にデプロイするための使いやすいコマンドが用意されています。

このチュートリアルでは、Elastic Beanstalk コンソールを使用して Composer のオプションを設定しました。この設定をアプリケーション出典のパートにするには、次のような設定ファイルを使用できます。

Example .ebextensions/composer.config

```
option_settings:
```

```
aws:elasticbeanstalk:container:php:phpini:
  document_root: /public
```

詳細については、「設定ファイル (.ebextensions) による高度な環境のカスタマイズ」を参照してください。

Symfony は、独自の設定ファイルを使用してデータベース接続を設定します。データベースと Symfony の接続手順については、「Symfony を使用してデータベースに接続する」を参照してください。

最後に、本稼働環境でアプリケーションを使用する予定の場合は、お客様の環境にカスタムドメイン名を設定し、安全な接続のために HTTPS を有効にすることが必要になります。

外部 Amazon RDS データベースを使用して高可用性の PHP アプリケーションを Elastic Beanstalk にデプロイする

このチュートリアルでは、外部の RDS DB インスタンスを起動し AWS Elastic Beanstalk、PHP アプリケーションを実行する高可用性環境を設定して接続するプロセスについて説明します。Elastic Beanstalk の外部で DB インスタンスを実行すると、データベースが環境のライフサイクルから切り離されます。これにより、複数の環境から同じデータベースに connect したり、あるデータベースから別のデータベースに切り替えたり、データベースに影響を与えずに Blue/Green デプロイを実行したりできます。

チュートリアルでは、ユーザー提供のテキストデータを保存するために MySQL データベースを使用するサンプル PHP アプリケーションを使用します。サンプルアプリケーションは、設定ファイルを使用して、PHP 設定を設定し、アプリケーションが使用するデータベース内にテーブルを作成します。Composer ファイルを使用して、デプロイの間にパッケージをインストールする方法も示します。

セクション

- 前提条件
- Amazon RDS に DB インスタンスを起動する
- Elastic Beanstalk 環境の作成
- セキュリティグループ、環境プロパティ、およびスケーリングの設定
- サンプルアプリケーションをデプロイする
- クリーンアップ
- 次のステップ

前提条件

スタートする前に、サンプルアプリケーション出典バンドルを GitHub からダウンロードします。eb-demo-php-simple-app-1.3.zip

Amazon Relational Database Service (Amazon RDS) タスクのためのこのチュートリアルの手順は、リソースをデフォルト Amazon Virtual Private Cloud (Amazon VPC) 内で起動していることを前提としています。すべての新しいアカウントでは、各リージョンにデフォルト VPC が含まれます。デフォルト VPC がない場合、手順は異なります。EC2-Classic およびカスタム VPC プラットフォームの手順については、「Amazon RDS で Elastic Beanstalk を使用する」を参照してください。

Amazon RDS に DB インスタンスを起動する

Elastic Beanstalk で実行されているアプリケーションで外部データベースを使用するには、まず Amazon RDS で DB インスタンスを起動します。Amazon RDS でインスタンスを起動すると、そのインスタンスは Elastic Beanstalk および Elastic Beanstalk 環境から完全に独立しているため、Elastic Beanstalk によって終了またはモニタリングされません。

Amazon RDS コンソールを使用して、マルチ AZ MySQL DB インスタンスを起動します。マルチ AZ 配置を選択すると、データベースがフェイルオーバーされ、出典 DB インスタンスがサービスを停止しても引き続き利用できます。

RDS DB インスタンスをデフォルト VPC 内で起動するには

1. RDS コンソールを開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. [データベースの作成] を選択します。
4. [Standard Create (スタンダード作成)] を選択します。

> **Important**
>
> [Easy Create (簡易作成)] を選択しないでください。これを選択した場合、この RDS DB の起動に必要な設定ができません。

5. [Additional configuration (追加の設定)] の [Initial database name (初期データベース名)] に「**ebdb**」と入力します。
6. デフォルト設定を確認し、特定の要件に従ってこれらの設定を調整します。以下のオプションに注目します。

- DB instance class (DB インスタンスクラス) – ワークロードに適したメモリ量と CPU 能力があるインスタンスサイズを選択します。
- Multi-AZ deployment (マルチ AZ 配置) – 高可用性を得るには、これを [Create an Aurora Replica/Reader node in a different AZ (異なる AZ に Aurora レプリカ/リーダーノードを作成)] に設定します。
- [Master username (マスター・ユーザーネーム)] と [Master password (マスターパスワード)] – データベースのユーザー名とパスワード。後で使用するため、これらの設定を書き留めておきます。

7. 残りのオプションのデフォルト設定を確認し、[データベースの作成] を選択します。

次に、DB インスタンスに添付するセキュリティグループを変更して、適切なポートへのインバウンドトラフィックを許可します。これは、後で Elastic Beanstalk 環境にアタッチするのと同じセキュリティグループで、追加するルールは、同じセキュリティグループ内の他のリソースにアクセス許可を付与するものです。

RDS インスタンスにアタッチされているセキュリティグループのインバウンドルールを変更するには

1. Amazon RDS コンソールを開きます。
2. [データベース] を選択します。
3. 詳細を表示する DB インスタンスの名前を選択します。
4. [Connectivity] (接続) セクションで、このページに表示される [Subnets] (サブネット)、[Security groups] (セキュリティグループ)、[Endpoint] (エンドポイント) をメモします。これは、後でこの情報を使用できるようにするためです。
5. [Security] (セキュリティ) には、DB インスタンスに関連付けられるセキュリティグループが表示されます。リンクを開いて、Amazon EC2 コンソールにセキュリティグループを表示します。
6. セキュリティグループの詳細で、インバウンド を選択します。
7. [編集] を選択します。
8. [ルールの追加] を選択します。
9. タイプ として、アプリケーションが使用する DB エンジンを選択します。
10. 出典 として、**sg-** と入力して、使用可能なセキュリティグループのリストを表示します。Elastic Beanstalk 環境で使用される Auto Scaling グループに関連付けられているセキュリ

ティグループを選択します。これは、環境内の Amazon EC2 インスタンスがデータベースにアクセスできるようにするためです。

11. [保存] を選択します。

DB インスタンスの作成には約 10 分かかります。その間に、Elastic Beanstalk 環境を作成します。

Elastic Beanstalk 環境の作成

Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境を作成します。[PHP] プラットフォームを選択し、デフォルト設定およびサンプルコードを受け入れます。環境を起動した後、データベースに connect するために環境を設定して、GitHub からダウンロードしたサンプルアプリケーションをデプロイできます。

環境を起動するには (コンソール)

1. 事前に設定されたリンク (console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced) を使用して、Elastic Beanstalk コンソールを開きます。
2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。
3. アプリケーションコード として、サンプルアプリケーション を選択します。
4. 確認と起動 を選択します。
5. 使用できるオプションを確認します。使用する有効なオプションを選択し、準備ができたら [アプリケーションの作成] を選択します。

環境の作成の所要時間は約 5 分です。以下のリソースが作成されます。

- EC2 インスタンス – 選択したプラットフォームでウェブ・ アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。

- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。
> Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

これらのリソースはすべて Elastic Beanstalk によって管理されます。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。起動した RDS DB インスタンスは、環境外にありますので、お客様の責任において、ライフサイクルを管理してください。

> **Note**
>
> Elastic Beanstalk が作成する Amazon S3 バケットは、環境間で共有され、環境の終了時に削除されません。詳細については、「Amazon S3 で Elastic Beanstalk を使用する」を参照してください。

セキュリティグループ、環境プロパティ、およびスケーリングの設定

DB インスタンスのセキュリティグループを実行中の環境に追加します。この手順によって、アタッチされる追加のセキュリティグループを使用して、Elastic Beanstalk が環境内のすべてのインスタンスの再プロビジョニングを行います。

環境にセキュリティグループを追加するには

- 次のいずれかを行います：
  - Elastic Beanstalk コンソールを使用してセキュリティグループを追加するには
    a. Elastic Beanstalk 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. ナビゲーションペインで、[環境] を選択し、リストから環境の名前を選択します。
    c. ナビゲーションペインで、[設定] を選択します。
    d. [インスタンス] 設定カテゴリで、[編集] を選択します。
    e. EC2 セキュリティグループで、Elastic Beanstalk が作成するインスタンスセキュリティグループに加えて、インスタンスにアタッチするセキュリティグループを選択します。
    f. ページの最下部で [適用] を選択し変更を保存します。
    g. 警告を読み取り、確認 を選択します。
  - 設定ファイルを使用してセキュリティグループを追加するには、`securitygroup-addexisting.config` サンプルファイルを使用します。

次に、環境プロパティを使用して環境に接続情報を渡します。サンプルアプリケーションは、環境内でデータベースをプロビジョニングする場合、Elastic Beanstalk 設定と一致するプロパティのデフォルトセットを使用します。

Amazon RDS DB インスタンスの環境プロパティを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [環境プロパティ] セクションで、アプリケーションが読み取りする変数を定義して、接続文字列を構成します。統合された RDS DB インスタンスがある環境との互換性を考慮して、以下の名前と値を使用します。パスワードを除くすべての値は、RDS コンソールで見つかります。

プロパティ名	説明	プロパティ値
`RDS_HOSTNAME`	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
`RDS_PORT`	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。
`RDS_DB_NAME`	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
`RDS_USERNAME`	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。

プロパティ名	説明	プロパティ値
RDS_PASSWORD	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

6. ページの最下部で [適用] を選択し変更を保存します。

最後に、より高いインスタンス数で、環境の Auto Scaling グループを設定します。環境のウェブサーバーが、単一障害点となることを防ぎ、サイトをサービス停止状態にせずに変更をデプロイすることが許可されるように、常に少なくとも 2 つのインスタンスを実行します。

高可用性のために環境の Auto Scaling グループを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [容量] 設定カテゴリで、[編集] を選択します。
5. [Auto Scaling group (Auto Scaling グループ)] セクションで、[Min instances (最小インスタンス数)] を **2** に設定します。
6. ページの最下部で [適用] を選択し変更を保存します。

サンプルアプリケーションをデプロイする

これで環境がサンプルアプリケーションを実行し、Amazon RDS に connect する準備ができました。環境にサンプルアプリケーションをデプロイします。

> **Note**
>
> GitHub から出典バンドルをダウンロードしていない場合は、ダウンロードします。eb-demo-php-simple-アプリケーション-1.3.zip

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

サイトはユーザーコメントを収集し、MySQL データベースを使用してデータを保存します。コメントを追加するには、[Share Your Thought] を選択し、コメントを入力してから [Submit Your Thought(Thought を送信)] を選択します。ウェブ・ アプリはデータベースにコメントを書き込みます。それにより、環境のすべてのインスタンスで読み取りができ、インスタンスがサービス停止状態になっても、失われないようにします。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

さらに、Elastic Beanstalk 環境の外に作成したデータベースリソースを終了できます。Amazon RDS DB インスタンスを終了する場合、スナップショットを作成し、後で別のインスタンスにデータを復元できます。

RDS DB インスタンスを終了するには

1. Amazon RDS コンソールを開きます。

2. [データベース] を選択します。
3. DB インスタンスを選択します。
4. [アクション] を選択し、[削除] を選択します。
5. スナップショットを作成するかどうかを選択してから、削除 を選択します。

次のステップ

アプリケーションの開発が進むにつれ、.zip ファイルを手動で作成して Elastic Beanstalk コンソールにアップロードすることなく、環境を管理してアプリケーションをデプロイする方法が必要になります。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) には、コマンドラインインターフェイスからアプリケーションを作成、設定して、Elastic Beanstalk 環境にデプロイするための使いやすいコマンドが用意されています。

サンプルアプリケーションは、設定ファイルを使用して、PHP 設定を設定し、存在しない場合はテーブルをデータベースに作成します。設定ファイルを使用して、環境の作成時に時間がかかる設定更新を避けるために、インスタンスのセキュリティグループ設定を指定できます。詳細については「設定ファイル (`.ebextensions`) による高度な環境のカスタマイズ」を参照してください。

開発とテストのために、マネージド DB インスタンスを環境に直接追加するために Elastic Beanstalk の機能を使用することが必要になる場合があります。環境内でのデータベースのセットアップについては、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

高パフォーマンスデータベースが必要な場合は、Amazon Aurora の使用を検討します。Amazon Aurora は MySQL 対応のデータベースエンジンで、商用データベースの特徴を低コストで提供します。別のデータベースにアプリケーションに connect するには、セキュリティグループ設定ステップを繰り返し、RDS 関連の環境プロパティを更新します。

最後に、本稼働環境でアプリケーションを使用する予定の場合は、お客様の環境にカスタムドメイン名を設定し、安全な接続のために HTTPS を有効にすることが必要になります。

外部 Amazon RDS データベースを使用して高可用性の WordPress ウェブサイトを Elastic Beanstalk にデプロイする

このチュートリアルでは、外部にある Amazon RDS DB インスタンスを起動する方法と AWS Elastic Beanstalk、WordPress ウェブサイトを実行する高可用性環境を設定して接続する方法について説明します。ウェブサイトでは、アップロードされたファイルの共有ストレージとして Amazon Elastic File System (Amazon EFS) を使用しています。

Elastic Beanstalk の外部で DB インスタンスを実行すると、データベースが環境のライフサイクルから切り離されます。これにより、複数の環境から同じデータベースに connect したり、あるデータベースから別のデータベースに切り替えたり、データベースに影響を与えずに Blue/Green デプロイを実行したりできます。

> **Note**
>
> WordPress のバージョンとの PHP リリースの互換性に関する最新情報については、WordPress のウェブサイトの PHP の互換性と WordPress のバージョンをご参照ください。WordPress 実装用の新しいリリースの PHP にアップグレードする前に、この情報をリファレンスしてください。

トピック

- 前提条件
- Amazon RDS に DB インスタンスを起動する
- WordPress のダウンロード
- Elastic Beanstalk 環境の起動
- セキュリティグループおよび環境プロパティの設定
- アプリケーションの設定とデプロイ
- WordPress のインストール
- キーとソルトの更新
- アクセス制限の削除
- Auto Scaling グループの設定
- WordPress のアップグレード
- クリーンアップ
- 次のステップ

前提条件

このチュートリアルでは、基本的な Elastic Beanstalk オペレーションと Elastic Beanstalk コンソールに関する知識があることを前提としています。まだ起動していない場合は、Elastic Beanstalk の使用を開始する方法について説明します。 の指示に従って、最初の Elastic Beanstalk 環境を起動します。

このガイドの手順に従うには、run command のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

デフォルト VPC

このチュートリアルの Amazon Relational Database Service (Amazon RDS) の手順では、リソースをデフォルト Amazon Virtual Private Cloud (Amazon VPC) を起動することを前提としています。すべての新しいアカウントには、各 AWS リージョンにデフォルトの VPC が含まれます。デフォルト VPC がない場合、手順は異なります。EC2-Classic およびカスタム VPC プラットフォームの手順については、「Amazon RDS で Elastic Beanstalk を使用する」を参照してください。

AWS リージョン

サンプルアプリケーションは、Amazon EFS をサポートする AWS リージョンでのみ機能する Amazon EFS を使用します。サポートされている AWS リージョンの詳細については、のAmazon Elastic File System のエンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。

Amazon RDS に DB インスタンスを起動する

Amazon RDS でインスタンスを起動すると、そのインスタンスは Elastic Beanstalk および Elastic Beanstalk 環境から完全に独立しているため、Elastic Beanstalk によって終了またはモニタリングされません。

以下のステップでは、Amazon RDS コンソールを使用して以下の操作を行います。

- MySQL エンジンを使用してデータベースを起動する。
- マルチ AZ 配置を有効にする。これにより、異なるアベイラビリティーゾーン (AZ) にスタンバイが作成されて、データの冗長性が得られ、I/O フリーズがなくなり、システムバックアップ中のレイテンシー・スパイクが最小限に抑えられます。

RDS DB インスタンスをデフォルト VPC 内で起動するには

1. RDS コンソールを開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. [データベースの作成] を選択します。
4. [Standard Create (スタンダード作成)] を選択します。

> **Important**
>
> [Easy Create (簡易作成)] を選択しないでください。これを選択した場合、この RDS DB の起動に必要な設定ができません。

5. [Additional configuration (追加の設定)] の [Initial database name (初期データベース名)] に「**ebdb**」と入力します。
6. デフォルト設定を確認し、特定の要件に従ってこれらの設定を調整します。以下のオプションに注目します。
   - DB instance class (DB インスタンスクラス) – ワークロードに適したメモリ量と CPU 能力があるインスタンスサイズを選択します。
   - Multi-AZ deployment (マルチ AZ 配置) – 高可用性を得るには、これを [Create an Aurora Replica/Reader node in a different AZ (異なる AZ に Aurora レプリカ/リーダーノードを作成)] に設定します。
   - [Master username (マスター・ユーザーネーム)] と [Master password (マスターパスワード)] – データベースのユーザー名とパスワード。後で使用するため、これらの設定を書き留めておきます。
7. 残りのオプションのデフォルト設定を確認し、[データベースの作成] を選択します。

DB インスタンスを作成したら、適切なポートでインバウンドトラフィックを許可するように、DB インスタンスに添付されたセキュリティグループを変更します。

> **Note**
>
> これは、後で Elastic Beanstalk 環境にアタッチするのと同じセキュリティグループであるため、ここで追加するルールにより、同じセキュリティグループ内の他のリソースに対するイングレスアクセス許可が付与されます。

RDS インスタンスにアタッチされているセキュリティグループのインバウンドルールを変更するには

1. Amazon RDS コンソールを開きます。
2. [データベース] を選択します。
3. 詳細を表示する DB インスタンスの名前を選択します。
4. [Connectivity] (接続) セクションで、このページに表示される [Subnets] (サブネット)、[Security groups] (セキュリティグループ)、[Endpoint] (エンドポイント) をメモします。これは、後でこの情報を使用できるようにするためです。
5. [Security] (セキュリティ) には、DB インスタンスに関連付けられるセキュリティグループが表示されます。リンクを開いて、Amazon EC2 コンソールにセキュリティグループを表示します。
6. セキュリティグループの詳細で、インバウンド を選択します。
7. [編集] を選択します。
8. [ルールの追加] を選択します。
9. タイプ として、アプリケーションが使用する DB エンジンを選択します。
10. 出典 として、**sg-** と入力して、使用可能なセキュリティグループのリストを表示します。Elastic Beanstalk 環境で使用される Auto Scaling グループに関連付けられているセキュリティグループを選択します。これは、環境内の Amazon EC2 インスタンスがデータベースにアクセスできるようにするためです。
11. [保存] を選択します。

DB インスタンスの作成には約 10 分かかります。その間、WordPress をダウンロードし、Elastic Beanstalk 環境を作成します。

WordPress のダウンロード

を使用して WordPress をデプロイする準備をするには AWS Elastic Beanstalk、WordPress ファイルをコンピュータにコピーし、正しい設定情報を提供する必要があります。

WordPress プロジェクトを作成するには

1. WordPress を wordpress.org からダウンロードします。

```
~$curl https://wordpress.org/wordpress-6.2.tar.gz -o wordpress.tar.gz
```

2. 設定ファイルを同じサンプルリポジトリからダウンロードします。

```
~$ wget https://github.com/aws-samples/eb-php-wordpress/releases/download/v1.1/eb-
php-wordpress-v1.zip
```

3. WordPress を抽出してフォルダ名を変更します。

```
~$ tar -xvf wordpress.tar.gz
~$ mv wordpress wordpress-beanstalk
~$ cd wordpress-beanstalk
```

4. インストールした WordPress で設定ファイルを抽出します。

```
~/wordpress-beanstalk$ unzip ../eb-php-wordpress-v1.zip
 creating: .ebextensions/
inflating: .ebextensions/dev.config
inflating: .ebextensions/efs-create.config
inflating: .ebextensions/efs-mount.config
inflating: .ebextensions/loadbalancer-sg.config
inflating: .ebextensions/wordpress.config
inflating: LICENSE
inflating: README.md
inflating: wp-config.php
```

Elastic Beanstalk 環境の起動

Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境を作成します。環境を起動した後、データベースに接続するように環境を設定して、WordPress コードを環境にデプロイできます。

以下のステップでは、Elastic Beanstalk コンソールを使用して以下の操作を行います。

- マネージド PHP プラットフォームを使用して、Elastic Beanstalk アプリケーションを作成します。
- デフォルト設定とサンプルコードをそのまま使用します。

環境を起動するには (コンソール)

1. 事前に設定されたリンク (console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced) を使用して、Elastic Beanstalk コンソールを開きます。

2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。
3. アプリケーションコード として、サンプルアプリケーション を選択します。
4. 確認と起動 を選択します。
5. 使用できるオプションを確認します。使用する有効なオプションを選択し、準備ができたら [アプリケーションの作成] を選択します。

環境の作成の所要時間は約 5 分です。以下のリソースが作成されます。

Elastic Beanstalk が作成したリソース

- EC2 インスタンス – 選択したプラットフォームでウェブ・アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。

- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。
> Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

これらのリソースはすべて Elastic Beanstalk によって管理されます。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

起動した Amazon RDS インスタンスは環境外にあるため、そのライフサイクルの管理はお客様の責任で行ってください。

> **Note**
>
> Elastic Beanstalk が作成する Amazon S3 バケットは、環境間で共有され、環境の終了時に削除されません。詳細については、「Amazon S3 で Elastic Beanstalk を使用する」を参照してください。

セキュリティグループおよび環境プロパティの設定

DB インスタンスのセキュリティグループを実行中の環境に追加します。この手順によって、アタッチされる追加のセキュリティグループを使用して、Elastic Beanstalk が環境内のすべてのインスタンスの再プロビジョニングを行います。

環境にセキュリティグループを追加するには

- 次のいずれかを行います：
  - Elastic Beanstalk コンソールを使用してセキュリティグループを追加するには
    a. Elastic Beanstalk 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. ナビゲーションペインで、[環境] を選択し、リストから環境の名前を選択します。
    c. ナビゲーションペインで、[設定] を選択します。
    d. [インスタンス] 設定カテゴリで、[編集] を選択します。
    e. EC2 セキュリティグループで、Elastic Beanstalk が作成するインスタンスセキュリティグループに加えて、インスタンスにアタッチするセキュリティグループを選択します。
    f. ページの最下部で [適用] を選択し変更を保存します。
    g. 警告を読み取り、確認 を選択します。
  - 設定ファイルを使用してセキュリティグループを追加するには、`securitygroup-addexisting.config` サンプルファイルを使用します。

次に、環境プロパティを使用して環境に接続情報を渡します。

WordPress アプリケーションは、環境内でデータベースをプロビジョニングする場合、Elastic Beanstalk 設定と一致するプロパティのデフォルトセットを使用します。

Amazon RDS DB インスタンスの環境プロパティを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

5. [環境プロパティ] セクションで、アプリケーションが読み取りする変数を定義して、接続文字列を構成します。統合された RDS DB インスタンスがある環境との互換性を考慮して、以下の名前と値を使用します。パスワードを除くすべての値は、RDS コンソールで見つかります。

プロパティ名	説明	プロパティ値
RDS_HOSTNAME	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
RDS_PORT	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。
RDS_DB_NAME	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
RDS_USERNAME	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。
RDS_PASSWORD	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

6. ページの最下部で [適用] を選択し変更を保存します。

アプリケーションの設定とデプロイ

以下に示すように、wordpress-beanstalk フォルダの構造が正しいことを確認してください。

```
wordpress-beanstalk$ tree -aL 1
```

```
.
### .ebextensions
### index.php
### LICENSE
### license.txt
### readme.html
### README.md
### wp-activate.php
### wp-admin
### wp-blog-header.php
### wp-comments-post.php
### wp-config.php
### wp-config-sample.php
### wp-content
### wp-cron.php
### wp-includes
### wp-links-opml.php
### wp-load.php
### wp-login.php
### wp-mail.php
### wp-settings.php
### wp-signup.php
### wp-trackback.php
### xmlrpc.php
```

プロジェクトリポジトリのカスタマイズした `wp-config.php` ファイルでは、前のステップで定義した環境変数を使用してデータベース接続を設定します。`.ebextensions` フォルダ内の設定ファイルでは、Elastic Beanstalk 環境内に追加のリソースを作成します。

設定ファイルをアカウントで使用するには、変更が必要です。ファイル内のプレースホルダ値を適切な ID に置き換えて、出典バンドルを作成します。

設定ファイルを更新して出典バンドルを作成するには

1. 設定ファイルを次のように変更します。

   - `.ebextensions/dev.config` – 環境へのアクセスを制限して、WordPress のインストールプロセス中に環境を保護します。ファイルの先頭にあるプレースホルダー IP アドレスを、お客様の環境の WordPress インストールウェブサイトへのアクセスに使用するコンピュータのパブリック IP アドレスに置き換えます。

> **Note**
> お客様のネットワークに応じて、IP アドレスブロックの使用が必要になる場合があります。

- `.ebextensions/efs-create.config` – VPC 内の各アベイラビリティーゾーン/サブネットに EFS ファイルシステムとマウントポイントを作成します。Amazon VPC コンソールで、デフォルトの VPC とサブネット ID を特定します。

2. プロジェクトフォルダのファイルを含む出典バンドルを作成します。次のコマンドでは、`wordpress-beanstalk.zip` という出典バンドルが作成されます。

```
~/eb-wordpress$ zip ../wordpress-beanstalk.zip -r * .[^.]*
```

ソースバンドルを Elastic Beanstalk にアップロードして、WordPress を環境にデプロイします。

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

WordPress のインストール

WordPress のインストールを完了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境 URL を選択して、ブラウザでサイトを開きます。サイトをまだ設定していないため、WordPress のインストールウィザードにリダイレクトされます。

4. スタンダードインストールを実行します。wp-config.php ファイルはすでに出典コードに存在し、環境からデータベース接続情報を読み取りするように設定されています。接続を設定するよう求められません。

インストールには約 1 分かかります。

キーとソルトの更新

WordPress 設定ファイル wp-config.php は、環境プロパティからキーとソルトの値も読み取ります。現在、これらのプロパティはすべて、test フォルダの wordpress.config ファイルによって .ebextensions に設定されています。

ハッシュソルトは環境プロパティの要件を満たす任意の値にすることができますが、出典コントロールには保存しないでください。環境でこれらのプロパティを直接設定するには、Elastic Beanstalk コンソールを使用します。

環境プロパティを更新するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[Configuration] を選択します。
4. [ソフトウェア] で、[編集] を選択します。
5. [Environment properties] で、以下のプロパティを変更します。
   - AUTH_KEY – AUTH_KEY に選択された値。
   - SECURE_AUTH_KEY – SECURE_AUTH_KEY に選択された値。
   - LOGGED_IN_KEY – LOGGED_IN_KEY に選択された値。
   - NONCE_KEY – NONCE_KEY に選択された値。
   - AUTH_SALT – AUTH_SALT に選択された値。
   - SECURE_AUTH_SALT – SECURE_AUTH_SALT に選択された値。
   - LOGGED_IN_SALT – LOGGED_IN_SALT に選択された値。
   - NONCE_SALT – NONCE_SALT に選択された値。
6. ページの最下部で [適用] を選択し変更を保存します。

> **Note**
>
> 環境のプロパティを直接設定すると、`wordpress.config` の値が直接上書きされます。

アクセス制限の削除

サンプルプロジェクトには、設定ファイル `loadbalancer-sg.config` が含まれています。このファイルによって、セキュリティグループが作成され、`dev.config` で設定した IP アドレスを使用して、そのグループが環境のロードバランサーに割り当てられます。これにより、ネットワークからのポート 80 を介した HTTP アクセスが制限されます。そうしないと、WordPress をインストールして管理者アカウントを設定する前に、部外者がサイトに connect する可能性があります。

WordPress のインストールが完了しました。次は、設定ファイルを削除し、サイトを公開します。

制限を削除し、環境を更新するには

1. プロジェクトディレクトリから `.ebextensions/loadbalancer-sg.config` ファイルを削除します。

```
~/wordpress-beanstalk$ rm .ebextensions/loadbalancer-sg.config
```

2. 出典バンドルを作成します。

```
~/eb-wordpress$ zip ../wordpress-beanstalk-v2.zip -r * .[^.]*
```

ソースバンドルを Elastic Beanstalk にアップロードして、WordPress を環境にデプロイします。

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

Auto Scaling グループの設定

最後に、より高いインスタンス数で、環境の Auto Scaling グループを設定します。少なくと 2 つのインスタンスを常に実行することで、環境のウェブ・サーバーが単一障害点になることを防ぎます。また、これにより、サイトを停止せずに変更をデプロイすることができます。

高可用性のために環境の Auto Scaling グループを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [容量] 設定カテゴリで、[編集] を選択します。
5. [Auto Scaling group (Auto Scaling グループ)] セクションで、[Min instances (最小インスタンス数)] を **2** に設定します。
6. ページの最下部で [適用] を選択し変更を保存します。

複数のインスタンスにわたるコンテンツのアップロードを support するために、サンプルプロジェクトでは Amazon EFS を使用して共有ファイルシステムを作成します。サイトで投稿を作成し、コンテンツをアップロードして共有ファイルシステムに保存します。投稿を表示して両方のインスタンスをヒットするまでページを複数回更新し、共有ファイルシステムが動作していることを検証します。

WordPress のアップグレード

新しいバージョンの WordPress にアップグレードするには、サイトをバックアップして新しい環境にデプロイします。

> **Important**
>
> WordPress 内の更新機能を使用したり、新しいバージョンを使用するように出典ファイルを更新しないでください。どちらのアクションでも、投稿 URL は (データベースとファイルシステムには残りますが) 404 エラーを返す場合があります。

WordPress をアップグレードするには

1. WordPress 管理者コンソールのエクスポートツールを使用して、投稿を XML ファイルにエクスポートします。

2. 旧バージョンのインストールに使用したのと同じ手順を使用して、新しいバージョンの WordPress を Elastic Beanstalk にデプロイしてインストールします。ダウンタイムを避けるために、新しいバージョンで新しい環境を作成できます。
3. 新しいバージョンで、WordPress インポートツールを管理者コンソールにインストールし、そのツールを使用して投稿が含まれている XML ファイルをインポートします。投稿が旧バージョンの管理者ユーザーによって作成されている場合は、管理者ユーザーをインポートしないで、投稿を新しいサイトの管理者ユーザーに割り当てます。
4. 新しいバージョンを別の環境にデプロイした場合は、CNAME スワップを実行して旧サイトのユーザーを新サイトにリダイレクトします。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

さらに、Elastic Beanstalk 環境の外に作成したデータベースリソースを終了できます。Amazon RDS DB インスタンスを終了する場合、スナップショットを作成し、後で別のインスタンスにデータを復元できます。

RDS DB インスタンスを終了するには

1. Amazon RDS コンソールを開きます。
2. [データベース] を選択します。
3. DB インスタンスを選択します。
4. [アクション] を選択し、[削除] を選択します。

5. スナップショットを作成するかどうかを選択してから、削除 を選択します。

次のステップ

アプリケーションの開発が進むにつれ、.zip ファイルを手動で作成して Elastic Beanstalk コンソールにアップロードすることなく、環境を管理してアプリケーションをデプロイする方法が必要になります。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) には、コマンドラインインターフェイスからアプリケーションを作成、設定して、Elastic Beanstalk 環境にデプロイするための使いやすいコマンドが用意されています。

サンプルアプリケーションは、設定ファイルを使用して PHP 設定を設定し、存在しない場合はテーブルをデータベースに作成します。設定ファイルを使用して、環境の作成時に時間がかかる設定更新を避けるために、インスタンスのセキュリティグループ設定を指定できます。詳細については「設定ファイル (`.ebextensions`) による高度な環境のカスタマイズ」を参照してください。

開発とテストのために、マネージド DB インスタンスを環境に直接追加するために Elastic Beanstalk の機能を使用することが必要になる場合があります。環境内でのデータベースのセットアップについては、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

高パフォーマンスデータベースが必要な場合は、Amazon Aurora の使用を検討します。Amazon Aurora は MySQL 対応のデータベースエンジンで、商用データベースの特徴を低コストで提供します。別のデータベースにアプリケーションに connect するには、セキュリティグループ設定ステップを繰り返し、RDS 関連の環境プロパティを更新します。

最後に、本稼働環境でアプリケーションを使用する予定の場合は、お客様の環境にカスタムドメイン名を設定し、安全な接続のために HTTPS を有効にすることが必要になります。

外部 Amazon RDS データベースを使用して高可用性の Drupal ウェブサイトを Elastic Beanstalk にデプロイする

このチュートリアルでは、外部の RDS DB インスタンスを起動するプロセスについて説明します AWS Elastic Beanstalk。そして、Drupal ウェブサイトを実行中の高可用性環境を設定して connect する方法を説明します。ウェブサイトでは、アップロードされたファイルの共有ストレージとして Amazon Elastic File System (Amazon EFS) を使用しています。Elastic Beanstalk の外部 DB インスタンスを実行すると、環境のライフサイクルからデータベースを分離し、複数の環境から同じデータベースに接続でき、あるデータベースから別のデータベースへ交換したり、データベースに影響を与えないで Blue-Green Deployment を実行できます。

セクション

- 前提条件
- Amazon RDS に DB インスタンスを起動する
- Elastic Beanstalk 環境の起動
- セキュリティ設定および環境プロパティの設定
- アプリケーションの設定とデプロイ
- Drupal のインストール
- Drupal 設定を更新してアクセス制限を削除する
- Auto Scaling グループの設定
- クリーンアップ
- 次のステップ

前提条件

このチュートリアルでは、基本的な Elastic Beanstalk オペレーションと Elastic Beanstalk コンソールに関する知識があることを前提としています。まだ起動していない場合は、Elastic Beanstalk の使用を開始する方法について説明します。 の指示に従って、最初の Elastic Beanstalk 環境を起動します。

このガイドの手順に従うには、run command のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

Amazon Relational Database Service (Amazon RDS) タスクのためのこのチュートリアルの手順は、リソースをデフォルト Amazon Virtual Private Cloud (Amazon VPC) 内で起動していることを前提としています。すべての新しいアカウントでは、各リージョンにデフォルト VPC が含まれます。デフォルト VPC がない場合、手順は異なります。EC2-Classic およびカスタム VPC プラットフォームの手順については、「Amazon RDS で Elastic Beanstalk を使用する」を参照してください。

サンプルアプリケーションは Amazon EFS を使用します。Amazon EFS をサポートする AWS リージョンでのみ機能します。 AWS リージョンのサポートについては、『』のAmazon Elastic File System のエンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。

Elastic Beanstalk 環境のプラットフォームで PHP 7.4 以前を使用している場合は、このチュートリアルでは Drupal バージョン 8.9.13 を使用することをお勧めします。PHP 8.0 以降を使用してインストールされたプラットフォームの場合は、Drupal 9.1.5 を使用することをお勧めします。

Drupal リリースとそれらが support する PHP バージョンの詳細については、Drupal のウェブサイトの PHP 要件をご参照ください。Drupal が推奨するコアバージョンは、ウェブサイト (https://www.drupal.org/project/drupal) に掲載されています。

Amazon RDS に DB インスタンスを起動する

Elastic Beanstalk で実行されているアプリケーションで外部データベースを使用するには、まず Amazon RDS で DB インスタンスを起動します。Amazon RDS でインスタンスを起動すると、そのインスタンスは Elastic Beanstalk および Elastic Beanstalk 環境から完全に独立しているため、Elastic Beanstalk によって終了またはモニタリングされません。

Amazon RDS コンソールを使用して、マルチ AZ MySQL DB インスタンスを起動します。マルチ AZ 配置を選択すると、データベースがフェイルオーバーされ、出典 DB インスタンスがサービスを停止しても引き続き利用できます。

RDS DB インスタンスをデフォルト VPC 内で起動するには

1. RDS コンソールを開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. [データベースの作成] を選択します。
4. [Standard Create (スタンダード作成)] を選択します。

> **Important**
>
> [Easy Create (簡易作成)] を選択しないでください。これを選択した場合、この RDS DB の起動に必要な設定ができません。

5. [Additional configuration (追加の設定)] の [Initial database name (初期データベース名)] に「**ebdb**」と入力します。
6. デフォルト設定を確認し、特定の要件に従ってこれらの設定を調整します。以下のオプションに注目します。

- DB instance class (DB インスタンスクラス) – ワークロードに適したメモリ量と CPU 能力があるインスタンスサイズを選択します。
- Multi-AZ deployment (マルチ AZ 配置) – 高可用性を得るには、これを [Create an Aurora Replica/Reader node in a different AZ (異なる AZ に Aurora レプリカ/リーダーノードを作成)] に設定します。
- [Master username (マスター・ユーザーネーム)] と [Master password (マスターパスワード)] – データベースのユーザー名とパスワード。後で使用するため、これらの設定を書き留めておきます。

7. 残りのオプションのデフォルト設定を確認し、[データベースの作成] を選択します。

次に、DB インスタンスに添付するセキュリティグループを変更して、適切なポートへのインバウンドトラフィックを許可します。これは、後で Elastic Beanstalk 環境にアタッチするのと同じセキュリティグループで、追加するルールは、同じセキュリティグループ内の他のリソースにアクセス許可を付与するものです。

RDS インスタンスにアタッチされているセキュリティグループのインバウンドルールを変更するには

1. Amazon RDS コンソールを開きます。
2. [データベース] を選択します。
3. 詳細を表示する DB インスタンスの名前を選択します。
4. [Connectivity] (接続) セクションで、このページに表示される [Subnets] (サブネット)、[Security groups] (セキュリティグループ)、[Endpoint] (エンドポイント) をメモします。これは、後でこの情報を使用できるようにするためです。
5. [Security] (セキュリティ) には、DB インスタンスに関連付けられるセキュリティグループが表示されます。リンクを開いて、Amazon EC2 コンソールにセキュリティグループを表示します。
6. セキュリティグループの詳細で、インバウンド を選択します。
7. [編集] を選択します。
8. [ルールの追加] を選択します。
9. タイプ として、アプリケーションが使用する DB エンジンを選択します。
10. 出典 として、**sg-** と入力して、使用可能なセキュリティグループのリストを表示します。Elastic Beanstalk 環境で使用される Auto Scaling グループに関連付けられているセキュリ

ティグループを選択します。これは、環境内の Amazon EC2 インスタンスがデータベースにアクセスできるようにするためです。

11. [保存] を選択します。

DB インスタンスの作成には約 10 分かかります。その間に、Elastic Beanstalk 環境を起動します。

Elastic Beanstalk 環境の起動

Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境を作成します。[PHP] プラットフォームを選択し、デフォルト設定およびサンプルコードを受け入れます。環境を起動した後、データベースに connect するように環境を設定して、Drupal コードを環境にデプロイできます。

環境を起動するには (コンソール)

1. 事前に設定されたリンク (console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced) を使用して、Elastic Beanstalk コンソールを開きます。
2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。
3. アプリケーションコード として、サンプルアプリケーション を選択します。
4. 確認と起動 を選択します。
5. 使用できるオプションを確認します。使用する有効なオプションを選択し、準備ができたら [アプリケーションの作成] を選択します。

環境の作成の所要時間は約 5 分です。以下のリソースが作成されます。

- EC2 インスタンス – 選択したプラットフォームでウェブ・ アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの

HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。

- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。
> Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

これらのリソースはすべて Elastic Beanstalk によって管理されます。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。起動した RDS DB インスタンスは、環境外にありますので、お客様の責任において、ライフサイクルを管理してください。

> **Note**
>
> Elastic Beanstalk が作成する Amazon S3 バケットは、環境間で共有され、環境の終了時に削除されません。詳細については、「Amazon S3 で Elastic Beanstalk を使用する」を参照してください。

セキュリティ設定および環境プロパティの設定

DB インスタンスのセキュリティグループを実行中の環境に追加します。この手順によって、アタッチされる追加のセキュリティグループを使用して、Elastic Beanstalk が環境内のすべてのインスタンスの再プロビジョニングを行います。

環境にセキュリティグループを追加するには

- 次のいずれかを行います：
  - Elastic Beanstalk コンソールを使用してセキュリティグループを追加するには
    a. Elastic Beanstalk 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. ナビゲーションペインで、[環境] を選択し、リストから環境の名前を選択します。
    c. ナビゲーションペインで、[設定] を選択します。
    d. [インスタンス] 設定カテゴリで、[編集] を選択します。
    e. EC2 セキュリティグループで、Elastic Beanstalk が作成するインスタンスセキュリティグループに加えて、インスタンスにアタッチするセキュリティグループを選択します。
    f. ページの最下部で [適用] を選択し変更を保存します。
    g. 警告を読み取り、確認 を選択します。
  - 設定ファイルを使用してセキュリティグループを追加するには、`securitygroup-addexisting.config` サンプルファイルを使用します。

次に、環境プロパティを使用して環境に接続情報を渡します。サンプルアプリケーションは、環境内でデータベースをプロビジョニングする場合、Elastic Beanstalk 設定と一致するプロパティのデフォルトセットを使用します。

Amazon RDS DB インスタンスの環境プロパティを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [環境プロパティ] セクションで、アプリケーションが読み取りする変数を定義して、接続文字列を構成します。統合された RDS DB インスタンスがある環境との互換性を考慮して、以下の名前と値を使用します。パスワードを除くすべての値は、RDS コンソールで見つかります。

プロパティ名	説明	プロパティ値
RDS_HOSTNAME	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
RDS_PORT	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。
RDS_DB_NAME	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
RDS_USERNAME	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。

プロパティ名	説明	プロパティ値
RDS_PASSWORD	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

6. ページの最下部で [適用] を選択し変更を保存します。

Drupal のインストール後に、SSH を使用してインスタンスに connect し、設定の詳細を取得する必要があります。SSH キーを環境のインスタンスに割り当てます。

SSH を設定するには

1. 作成済みのキーペアがない場合は、Amazon EC2 コンソールのキーペアのページを開き、手順に従って作成します。
2. Elastic Beanstalk 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。
3. ナビゲーションペインで、[環境] を選択し、リストから環境の名前を選択します。
4. ナビゲーションペインで、[設定] を選択します。
5. [セキュリティ] で、[編集] を選択します。
6. EC2 キーペア について、キーペアを選択します。
7. ページの最下部で [適用] を選択し変更を保存します。

アプリケーションの設定とデプロイ

Elastic Beanstalk の Drupal プロジェクトを作成するには、Drupal ソースコードをダウンロードして、このコードを GitHub の aws-samples/eb-php-drupal リポジトリにあるファイルに結合します。

Drupal プロジェクトを作成するには

1. 次のコマンドを実行して、www.drupal.org/download から Drupal をダウンロードします。ダウンロードの詳細については、Drupal のウェブサイトをご参照ください。

   Elastic Beanstalk 環境のプラットフォームで PHP 7.4 以前を使用している場合は、このチュートリアルでは Drupal バージョン 8.9.13 をダウンロードすることをお勧めします。次のコマンドを実行してダウンロードできます。

```
~$ curl https://ftp.drupal.org/files/projects/drupal-8.9.13.tar.gz -o drupal.tar.gz
```

プラットフォームが PHP 8.0 以降を使用している場合は、Drupal 9.1.5 をダウンロードすることをお勧めします。このコマンドを使用してダウンロードできます。

```
~$ curl https://ftp.drupal.org/files/projects/drupal-9.1.5.tar.gz -o drupal.tar.gz
```

Drupal リリースとそれらが support する PHP バージョンの詳細については、Drupal の公式ドキュメントの PHP 要件をご参照ください。Drupal が推奨するコアバージョンは、Drupal のウェブサイトに掲載されています。

2. 次のコマンドを使用して、サンプルリポジトリから設定ファイルをダウンロードします。

```
~$ wget https://github.com/aws-samples/eb-php-drupal/releases/download/v1.1/eb-php-drupal-v1.zip
```

3. Drupal を抽出してフォルダ名を変更します。

   Drupal 8.9.13 をダウンロードした場合:

```
~$ tar -xvf drupal.tar.gz
~$ mv drupal-8.9.13 drupal-beanstalk
~$ cd drupal-beanstalk
```

   Drupal 9.1.5 をダウンロードした場合:

```
~$ tar -xvf drupal.tar.gz
~$ mv drupal-9.1.5 drupal-beanstalk
~$ cd drupal-beanstalk
```

4. インストールした Drupal で設定ファイルを抽出します。

```
~/drupal-beanstalk$ unzip ../eb-php-drupal-v1.zip
 creating: .ebextensions/
 inflating: .ebextensions/dev.config
 inflating: .ebextensions/drupal.config
 inflating: .ebextensions/efs-create.config
 inflating: .ebextensions/efs-filesystem.template
 inflating: .ebextensions/efs-mount.config
 inflating: .ebextensions/loadbalancer-sg.config
```

```
  inflating: LICENSE
  inflating: README.md
  inflating: beanstalk-settings.php
```

以下に示すように、drupal-beanstalk フォルダの構造が正しいことを確認してください。

```
drupal-beanstalk$ tree -aL 1
.
### autoload.php
### beanstalk-settings.php
### composer.json
### composer.lock
### core
### .csslintrc
### .ebextensions
### .ebextensions
### .editorconfig
### .eslintignore
### .eslintrc.json
### example.gitignore
### .gitattributes
### .htaccess
### .ht.router.php
### index.php
### LICENSE
### LICENSE.txt
### modules
### profiles
### README.md
### README.txt
### robots.txt
### sites
### themes
### update.php
### vendor
### web.config
```

プロジェクトリポジトリの beanstalk-settings.php ファイルでは、前のステップで定義した環境変数を使用してデータベース接続を設定します。.ebextensions フォルダ内の設定ファイルでは、Elastic Beanstalk 環境内に追加のリソースを作成します。

設定ファイルをアカウントで使用するには、変更が必要です。ファイル内のプレースホルダ値を適切な ID に置き換えて、出典バンドルを作成します。

設定ファイルを更新して出典バンドルを作成するには

1. 設定ファイルを次のように変更します。
   - `.ebextensions/dev.config` – 環境へのアクセスを IP アドレスに制限して、Drupal のインストールプロセス中に環境を保護します。ファイルの先頭にあるプレースホルダー IP アドレスをパブリック IP アドレスに置き換えます。
   - `.ebextensions/efs-create.config` – VPC 内の各アベイラビリティーゾーン/サブネットに EFS ファイルシステムとマウントポイントを作成します。Amazon VPC コンソールで、デフォルトの VPC とサブネット ID を特定します。
2. プロジェクトフォルダのファイルを含む出典バンドルを作成します。次のコマンドでは、`drupal-beanstalk.zip` という出典バンドルが作成されます。`vendor` フォルダ内のファイルは除外されます。これらのファイルは、多くのスペースを使用するだけでなく、アプリケーションを Elastic Beanstalk にデプロイするのに不要です。

```
~/eb-drupal$ zip ../drupal-beanstalk.zip -r * .[^.]* -x "vendor/*"
```

ソースバンドルを Elastic Beanstalk にアップロードして、Drupal を環境にデプロイします。

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

Drupal のインストール

Drupal のインストールを完了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境 URL を選択して、ブラウザでサイトを開きます。サイトがまだ設定されていないため、Drupal のインストールウィザードにリダイレクトされます。
4. データベースの次の設定でスタンダードインストールを実行します。
   - データベース名 – Amazon RDS コンソールに表示される [DB Name (DB 名)]。
   - データベースのユーザーネームとパスワード – データベースの作成時に入力した [Master Username (マスターユーザー名)] と [Master Password (マスターパスワード)] の値。
   - アドバンストオプション > ホスト – Amazon RDS コンソールに表示される DB インスタンスの [Endpoint (エンドポイント)]。

インストールには約 1 分かかります。

Drupal 設定を更新してアクセス制限を削除する

Drupal のインストールプロセスで、インスタンスの `sites/default` フォルダに `settings.php` というファイルを作成しました。このファイルを出典コードで使用して、以降のデプロイでサイトがリセットされるのに防ぐ必要がありますが、ファイルには現在ソースにコミットしたくないシークレットが含まれています。アプリケーションインスタンスに connect して設定ファイルから情報を取得します。

SSH を使用してアプリケーションインスタンスに connect するには

1. Amazon EC2 コンソールのインスタンスページを開きます。
2. アプリケーションインスタンスを選択します。これは、Elastic Beanstalk 環境と同じ名前を付けたインスタンスです。
3. [Connect] を選択します。
4. 手順に従ってインスタンスを SSH に connect します。コマンドは次のようになります。

```
$ ssh -i ~/.ssh/mykey ec2-user@ec2-00-55-33-222.us-west-2.compute.amazonaws.com
```

設定ファイルの最後の行から同期ディレクトリ ID を取得します。

```
[ec2-user ~]$ tail -n 1 /var/app/current/sites/default/settings.php
```

```
$config_directories['sync'] = 'sites/default/files/
config_4ccfX2sPQm79p1mk5IbUq9S_FokcENO4mxyC-L18-4g_xKj_7j9ydn31kDOYOgnzMu071Tvc4Q/
sync';
```

ファイルにはサイトの現在のハッシュキーも含まれていますが、現在の値は無視して独自の値を使用できます。

同期ディレクトリパスとハッシュキーを環境プロパティに割り当てます。プロジェクトリポジトリのカスタマイズされた設定ファイルは、前に設定したデータベース接続プロパティに加えて、これらのプロパティを読み取ることで、デプロイ中にサイトを設定します。

Drupal 設定プロパティ

- `SYNC_DIR` – 同期ディレクトリへのパス。
- `HASH_SALT` – 環境プロパティの要件を満たす任意の文字列値。

Elastic Beanstalk コンソールで環境変数を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. Runtime 環境変数まで下にスクロールします。
6. 環境変数の追加 を選択します。
7. ソース プレーンテキストを選択します。

> **Note**
>
> ドロップダウンの Secrets Manager と SSM パラメータストアの値は、認証情報や API キーなどの機密データを保存するシークレットとして環境変数を設定するためのものです。詳細については、「AWS Secrets Manager および Parameter Store での Elastic Beanstalk AWS Systems Manager の使用」を参照してください。

8. 環境変数名と環境変数値のペアを入力します。
9. さらに変数を追加する必要がある場合は、ステップ 6 からステップ 8 を繰り返します。
10. ページの最下部で [適用] を選択し変更を保存します。

最後に、サンプルプロジェクトに含まれている設定ファイル (`loadbalancer-sg.config`) では、セキュリティグループを作成して、これを環境のロードバランサーに割り当てます。そのために、`dev.config` で設定した IP アドレスを使用して、ポート 80 経由でネットワークからの接続への HTTP アクセスを制限します。そうしないと、Drupal をインストールして管理者アカウントを設定する前に、部外者がサイトに connect する可能性があります。

Drupal の設定を更新してアクセス制限を削除するには

1. プロジェクトディレクトリから `.ebextensions/loadbalancer-sg.config` ファイルを削除します。

```
~/drupal-beanstalk$ rm .ebextensions/loadbalancer-sg.config
```

2. カスタマイズした `settings.php` ファイルをサイトのフォルダ内にコピーします。

```
~/drupal-beanstalk$ cp beanstalk-settings.php sites/default/settings.php
```

3. 出典バンドルを作成します。

```
~/eb-drupal$ zip ../drupal-beanstalk-v2.zip -r * .[^.]* -x "vendor/*"
```

ソースバンドルを Elastic Beanstalk にアップロードして、Drupal を環境にデプロイします。

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

Auto Scaling グループの設定

最後に、より高いインスタンス数で、環境の Auto Scaling グループを設定します。環境のウェブサーバーが、単一障害点となることを防ぎ、サイトをサービス停止状態にせずに変更をデプロイすることが許可されるように、常に少なくとも 2 つのインスタンスを実行します。

高可用性のために環境の Auto Scaling グループを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [容量] 設定カテゴリで、[編集] を選択します。
5. [Auto Scaling group (Auto Scaling グループ)] セクションで、[Min instances (最小インスタンス数)] を **2** に設定します。
6. ページの最下部で [適用] を選択し変更を保存します。

複数のインスタンスにわたるコンテンツのアップロードを support するために、サンプルプロジェクトでは Amazon Elastic File System を使用して共有ファイルシステムを作成します。サイトで投稿を作成し、コンテンツをアップロードして共有ファイルシステムに保存します。投稿を表示して両方のインスタンスをヒットするまでページを複数回更新し、共有ファイルシステムが動作していることを検証します。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

さらに、Elastic Beanstalk 環境の外に作成したデータベースリソースを終了できます。Amazon RDS DB インスタンスを終了する場合、スナップショットを作成し、後で別のインスタンスにデータを復元できます。

RDS DB インスタンスを終了するには

1. Amazon RDS コンソールを開きます。
2. [データベース] を選択します。
3. DB インスタンスを選択します。
4. [アクション] を選択し、[削除] を選択します。
5. スナップショットを作成するかどうかを選択してから、削除 を選択します。

次のステップ

アプリケーションの開発が進むにつれ、.zip ファイルを手動で作成して Elastic Beanstalk コンソールにアップロードすることなく、環境を管理してアプリケーションをデプロイする方法が必要になります。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) には、コマンドラインインターフェイスからアプリケーションを作成、設定して、Elastic Beanstalk 環境にデプロイするための使いやすいコマンドが用意されています。

サンプルアプリケーションは、設定ファイルを使用して、PHP 設定を設定し、存在しない場合はテーブルをデータベースに作成します。設定ファイルを使用して、環境の作成時に時間がかかる設定更新を避けるために、インスタンスのセキュリティグループ設定を設定できます。詳細については「設定ファイル (`.ebextensions`) による高度な環境のカスタマイズ」を参照してください。

開発とテストのために、マネージド DB インスタンスを環境に直接追加するために Elastic Beanstalk の機能を使用することが必要になる場合があります。環境内でのデータベースのセットアップについては、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

高パフォーマンスデータベースが必要な場合は、Amazon Aurora の使用を検討します。Amazon Aurora は MySQL 対応のデータベースエンジンで、商用データベースの特徴を低コストで提供します。別のデータベースにアプリケーションに connect するには、セキュリティグループ設定ステップを繰り返し、RDS 関連の環境プロパティを更新します。

最後に、本稼働環境でアプリケーションを使用する予定の場合は、お客様の環境にカスタムドメイン名を設定し、安全な接続のために HTTPS を有効にすることが必要になります。

Python アプリケーションの Elastic Beanstalk でのデプロイ

この章では、Python ウェブアプリケーションを設定してデプロイする手順について説明します AWS Elastic Beanstalk。Elastic Beanstalk を使用すると、Amazon Web Services を使用して簡単に Python ウェブアプリケーションのデプロイ、管理、スケーリングができます。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) または Elastic Beanstalk コンソールを使用すると、わずか数分でアプリケーションをデプロイできます。Elastic Beanstalk アプリケーションをデプロイした後も、EB CLI を使用してアプリケーションと環境を管理することも、Elastic Beanstalk コンソール AWS CLIまたは APIs を使用することもできます。

EB CLI を使用して Python Hello World ウェブアプリケーションを作成してデプロイするには、「Python の QuickStart」のステップバイステップの手順に従います。

トピック

- QuickStart: Elastic Beanstalk に Python アプリケーションをデプロイする
- Elastic Beanstalk 用の Python 開発環境の設定
- Elastic Beanstalk Python プラットフォームを使用する
- Elastic Beanstalk への Flask アプリケーションのデプロイ
- Elastic Beanstalk への Django アプリケーションのデプロイ
- Amazon RDS DB インスタンスを Python Elastic Beanstalk 環境に追加する
- Python のツールとリソース

QuickStart: Elastic Beanstalk に Python アプリケーションをデプロイする

この QuickStart チュートリアルでは、Python アプリケーションを作成し、 AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

> **Note**
>
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにはアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: Python アプリケーションを作成する
- ステップ 2: アプリケーションをローカルに実行する
- ステップ 3: EB CLI を使用して Python アプリケーションをデプロイする
- ステップ 4: Elastic Beanstalk でアプリケーションを実行する
- ステップ 5：クリーンアップ
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

AWS アカウントをすでにお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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、通話呼び出し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するように求められます。

   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、 のセキュリティを確保し AWS IAM Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 「 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

EB CLI

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

Python および Flask フレームワーク

次のコマンドを実行して、pip がインストールされ動作している Python バージョンがあることを確認します。

```
~$ python3 --version
Python 3.N.N
>~$ python3 -m pip --version
pip X.Y.Z from ... (python 3.N.N)
```

前のコマンドのいずれかが「Python が見つからない」と返す場合は、python3 ではなく python を使用する次のコマンドを実行します。エイリアスとシンボリックリンクの設定は、オペレーティングシステムと個々のカスタマイズによって異なる可能性があるため、python3 コマンドはマシンで機能しない場合があります。

```
~$ python --version
Python 3.N.N
>~$ python -m pip --version
pip X.Y.Z from ... (python 3.N.N)
```

ローカルマシンに Python がインストールされていない場合は、Python ウェブサイトの「Python ダウンロード」ページからダウンロードできます。Elastic Beanstalk がサポートする Python 言語バージョンのリストについては、「AWS Elastic Beanstalk プラットフォーム」ガイドの「サポートされている Python プラットフォーム」を参照してください。Python ダウンロードウェブサイトには、「Python デベロッパーガイド」へのリンクがあり、インストールと設定の手順が記載されています。

> **Note**
>
> Python pip パッケージは、デフォルトで Python 3.4 以降に含まれています。

サポートされているバージョンの Python があるが、pip ではないことが出力で示された場合は、pip.pypa.io ウェブサイトの「インストール」ページを参照してください。pip がない Python 環境に pip をインストールするためのガイダンスがここに記載されています。

次のコマンドを実行して、Flask がインストールされているかどうかを確認します。

```
~$ pip list | grep Flask
```

Flask がインストールされていない場合は、次のコマンドを実行してインストールできます。

```
~$ pip install Flask
```

ステップ 1: Python アプリケーションを作成する

プロジェクトディレクトリを作成します。

```
~$ mkdir eb-python
~$ cd eb-python
```

「Hello Elastic Beanstalk!」というサンプルを作成します。Elastic Beanstalk を使用してデプロイする Python アプリケーション。

作成したばかりのディレクトリに `application.py` という名前のテキストファイルを次の内容で作成します。

Example **`~/eb-python/application.py`**

```
from flask import Flask
application = Flask(__name__)

@application.route('/')
def hello_elastic_beanstalk():
        return 'Hello Elastic Beanstalk!'
```

次の行がある `requirements.txt` という名前のテキストファイルを作成します。このファイルには、アプリケーションを実行するために必要な `pip` パッケージが含まれています。

Example **`~/eb-python/requirements.txt`**

```
Flask
```

ステップ 2: アプリケーションをローカルに実行する

アプリケーションをローカルで実行するには、次のコマンドを実行します。

```
~/eb-python$ export FLASK_APP=application.py && flask run --port 5000
```

次のような出力が表示されます。

```
Serving Flask app 'application.py'
Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a
 production WSGI server instead.
Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [01/Jan/1970 00:00:00] "GET / HTTP/1.1" 200 -
```

ウェブブラウザで `http://localhost:5000` にアクセスします。ウェブブラウザに「Hello Elastic Beanstalk!」と表示されます。

ステップ 3: EB CLI を使用して Python アプリケーションをデプロイする

次のコマンドを実行して、このアプリケーションの Elastic Beanstalk 環境を作成します。

環境を作成し、Python アプリケーションをデプロイするには

1. eb init コマンドを使用して EB CLI リポジトリを初期化します。

   ```
   ~/eb-python$ eb init -p python-3.9 python-tutorial --region us-east-2
   ```

   このコマンドは、`python-tutorial` という名前のアプリケーションを作成し、ローカルリポジトリを設定して指定された Python プラットフォームバージョンで環境を作成します。

2. (オプション) eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH を使用して connect できるようにします。

   ```
   ~/eb-python$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

   1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従ってキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。

3. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。Elastic Beanstalk は、アプリケーションの zip ファイルを自動的にビルドし、ポート 5000 で起動します。

```
~/eb-python$ eb create python-env
```

Elastic Beanstalk が環境を作成するのに約 5 分かかります。

ステップ 4: Elastic Beanstalk でアプリケーションを実行する

環境を作成するプロセスが完了したら、eb open でウェブサイトを開きます。

```
~/eb-python$ eb open
```

お疲れ様でした。Elastic Beanstalk で Python アプリケーションをデプロイしました。これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ Window が開きます。

ステップ 5：クリーンアップ

アプリケーションでの作業が終了したら、環境を終了できます。Elastic Beanstalk は、環境に関連付けられているすべての AWS リソースを終了します。

EB CLI を使用して Elastic Beanstalk 環境を終了するには、次のコマンドを実行します。

```
~/eb-python$ eb terminate
```

AWS アプリケーションの リソース

1 つのインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありません。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリ

プトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。

- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

> **その他のチュートリアルを試す**
>
> 異なるサンプルアプリケーションで他のチュートリアルを試したい場合は、以下のチュートリアルを参照してください。
>
> - Elastic Beanstalk への Flask アプリケーションのデプロイ

- Elastic Beanstalk への Django アプリケーションのデプロイ

1 つか 2 つのサンプルアプリケーションをデプロイし、ローカルで Python アプリケーションを開発して実行する準備が整ったら、「Elastic Beanstalk 用の Python 開発環境の設定」を参照します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。

Elastic Beanstalk 用の Python 開発環境の設定

このトピックでは、Python 開発環境を設定し、アプリケーションを AWS Elastic Beanstalkにデプロイする前にローカルでテストする手順について説明します。また、便利なツールのインストール手順を提供するウェブサイトも参照します。

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

セクション

- 前提条件
- 仮想環境の使用
- Elastic Beanstalk 用の Python プロジェクトの設定

前提条件

次のリストは、Elastic Beanstalk と Python アプリケーションを使用するための一般的な前提条件を示しています。

- Python 言語 – 選択した Elastic Beanstalk Python プラットフォームバージョンに含まれている Python 言語のバージョンをインストールします。サポートされている Python 言語バージョンのリストについては、「AWS Elastic Beanstalk プラットフォームガイド」の「サポートされている Python プラットフォーム」を参照してください。開発マシンに Python がまだ設定されていない場合は、Python ウェブサイトの「Python ダウンロード」ページを参照してください。
- **pip** ユーティリティ – pip ユーティリティは Python のパッケージインストーラです。アプリケーション環境の設定方法を Elastic Beanstalk が判断できるように、プロジェクトの依存関係をインストールし一覧表示します。pip の詳細については、pip.pypa.io ウェブサイトの pip ページを参照してください。
- (オプション) Elastic Beanstalk コマンドラインインターフェイス (EB CLI) – EB CLI は、必要なデプロイファイルを使用してアプリケーションをパッケージ化できます。また、Elastic Beanstalk 環境を作成して、そこにアプリケーションをデプロイすることもできます。Elastic Beanstalk コンソールからもデプロイできます。そのため、EB CLI が必ず必要というわけではありません。
- 作業用 **SSH** のインストール – SSH プロトコルを使用して実行中のインスタンスに接続し、デプロイを検査またはデバッグできます。
- **virtualenv** パッケージ – この virtualenv ツールは、アプリケーションの開発およびテスト環境を作成します。Elastic Beanstalk は、アプリケーションで不要な追加のパッケージをインストールすることなく、この環境をレプリケートできます。詳細については、virtualenv のウェブサイトを参照してください。Python をインストールしたら、次のコマンドを使用して virtualenv パッケージをインストールできます。

```
$ pip install virtualenv
```

仮想環境の使用

前提条件をインストールしたら、virtualenv を使用して仮想環境をセットアップし、アプリケーションの依存関係をインストールします。仮想環境を使用することで、アプリケーションで必要なパッケージを正確に識別し、アプリケーションを実行する EC2 インスタンスにそれらの必要なパッケージをインストールすることができます。

仮想環境をセットアップするには

1. コマンドライン・ウィンドウを開き、次のように入力します。

```
$ virtualenv /tmp/eb_python_app
```

eb_python_app を、アプリケーションに対応するわかりやすい名前に置き換えます (アプリケーション名を使用することをお勧めします)。virtualenv コマンドによって指定したディレクトリに仮想環境が作成され、アクションの結果が出力されます。

```
Running virtualenv with interpreter /usr/bin/python
New python executable in /tmp/eb_python_app/bin/python3.12
Also creating executable in /tmp/eb_python_app/bin/python
Installing setuptools, pip...done.
```

2. 仮想環境の準備が整ったら、環境の activate ディレクトリにある bin スクリプトを実行してスタートします。たとえば、前のステップで作成した eb_python_app 環境をスタートするには、次のように入力します。

```
$ source /tmp/eb_python_app/bin/activate
```

コマンドプロンプトの冒頭には毎回、仮想環境の名前 (例: (eb_python_app)) が出力され、仮想 Python 環境を使用していることがわかるようになっています。

3. 仮想環境の使用を停止し、インストールされているすべてのライブラリを含むシステムのデフォルトの Python インタプリタに戻るには、deactivate コマンドを実行します。

```
(eb_python_app) $ deactivate
```

> **Note**
> 仮想環境の作成後は、activate スクリプトを再び実行することで、いつでも再起動できます。

Elastic Beanstalk 用の Python プロジェクトの設定

Elastic Beanstalk CLI を使用して、Elastic Beanstalk でのデプロイ用に Python アプリケーションを準備します。

Elastic Beanstalk でのデプロイ用に Python アプリケーションを設定するには

1. 仮想環境で、プロジェクトのディレクトリツリーの最上位 (python_eb_app) に戻り、次のように入力します。

```
pip freeze >requirements.txt
```

このコマンドは、仮想環境にインストールされているパッケージの名前とバージョンを `requirements.txt` にコピーします。たとえば、PyYAML パッケージ、バージョン 3.11 が仮想環境にインストールされている場合、このファイルには次の行が含まれます。

```
PyYAML==3.11
```

これにより、Elastic Beanstalk は、アプリケーションの開発とテストに使用されたのと同じパッケージとバージョンを使用して、アプリケーションの Python 環境をレプリケートすることができます。

2. eb init コマンドで EB CLI リポジトリを設定します。画面の指示に従ってリージョン、プラットフォーム、その他のオプションを選択します。

デフォルトでは、Elastic Beanstalk は、アプリケーションを開始するために `application.py` というファイルを探します。作成した Python プロジェクトにこのファイルが存在しない場合は、アプリケーション環境の調整が必要になります。また、アプリケーションのモジュールをロードできるように、環境変数を設定する必要があります。詳細については「Elastic Beanstalk Python プラットフォームを使用する」を参照してください。

Elastic Beanstalk Python プラットフォームを使用する

このトピックでは、Elastic Beanstalk で Python アプリケーションを設定、ビルド、実行する方法について説明します。

AWS Elastic Beanstalk は、Python プログラミング言語のさまざまなバージョンで多数のプラットフォームブランチをサポートしています。完全なリストについては、「AWS Elastic Beanstalk プラットフォーム」ドキュメントの「Python」を参照してください。

Python ウェブアプリケーションは、WSGI を使用してプロキシサーバーの背後で実行できます。Elastic Beanstalk には、デフォルトの WSGI サーバーとして Gunicorn が用意されています。

ソ出典バンドルに `Procfile` を追加して、アプリケーションの WSGI サーバーを指定および設定することができます。詳細については、「the section called “[Procfile]”」を参照してください

Pipenv によって作成された `Pipfile` ファイルと `Pipfile.lock` ファイルを使用して、Python パッケージの依存関係やその他の要件を指定できます。依存関係の指定の詳細については、「the section called “依存関係の指定”」を参照してください。

Elastic Beanstalk には、Elastic Beanstalk 環境内の EC2 インスタンスで実行されるソフトウェアのカスタマイズに使用できる設定オプションが用意されています。アプリケーションに必要な環境変数を設定し、Amazon S3 に対してログのローテーションを有効にしたら、アプリケーションの出典で静的ファイルが含まれるフォルダを、プロキシサーバーによって提供されるパスにマッピングできます。

設定オプションは実行中の環境の設定を変更するために Elastic Beanstalk コンソールで利用できます。環境を終了したときにその設定が失われないようにするため、保存済み設定を使用して設定を保存し、それを後で他の環境に適用することができます。

ソースコードの設定を保存する場合、設定ファイルを含めることができます。設定ファイルの設定は、環境を作成するたびに、またはアプリケーションをデプロイするたびに適用されます。設定ファイルを使用して、デプロイの間にパッケージをインストールしたり、スクリプトを実行したり、他のインスタンスのカスタマイズオペレーションを実行することもできます。

Elastic Beanstalk コンソールで適用される設定は、設定ファイルに同じ設定があれば、それらの設定を上書きします。これにより、設定ファイルでデフォルト設定を定義し、コンソールでそのデフォルト設定を環境固有の設定で上書きできます。設定の優先順位の詳細と設定の他の変更方法については、「設定オプション」を参照してください。

`pip` から入手可能な Python パッケージの場合、アプリケーションの出典コードのルートに要件ファイルを含めることができます。Elastic Beanstalk は、デプロイ時に要件ファイルで指定された依存関係パッケージをすべてインストールします。詳細については、「the section called “依存関係の指定”」を参照してください

Elastic Beanstalk Linux ベースのプラットフォームを拡張するさまざまな方法の詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

Python 環境の設定

Python プラットフォーム設定では、Amazon EC2 インスタンスの動作を微調整できます。Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境の Amazon EC2 インスタンス設定を編集できます。

Elastic Beanstalk コンソールを使用して、Python プロセス設定の設定、Amazon S3 への AWS X-Rayログローテーションの有効化、およびアプリケーションが環境から読み取ることができる変数の設定を行います。

Elastic Beanstalk コンソールで Python 環境を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

Python 設定

- [プロキシサーバー] – 環境インスタンスで使用するプロキシサーバーです。デフォルトでは、nginx が使用されます。
- [WSGI パス] – メインアプリケーション· ファイルの名前またはパス。たとえば、`application.py`、`django/wsgi.py` などです。
- [NumProcesses] – 各アプリケーションインスタンスで実行するプロセスの数。
- [NumThreads] – 各プロセスで実行するスレッドの数。

AWS X-Ray 設定

- X-Ray デーモン – AWS X-Ray デーモンを実行して、 からのトレースデータを処理しますAWS X-Ray SDK for Python。

ログオプション

[ログ Options] セクションには、2 つの設定があります。

- [Instance profile] – アプリケーションに関連付けられた Amazon S3 バケットへのアクセス許可が付与されているインスタンスプロファイルを指定します。
- [Enable log file rotation to Amazon S3] (Amazon S3 へのログファイルのローテーションの有効化) - アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーするかどうかを指定します。

静的ファイル

パフォーマンスを向上させるために、[Static files] (静的ファイル) セクションを使用して、ウェブアプリケーション内のディレクトリセットから静的ファイル (HTML、イメージなど) を配信するようにプロキシサーバーを設定することができます。ディレクトリごとに、仮想パスをディレクトリマッピングに設定します。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接　　処理します。

設定ファイルまたは、Elastic Beanstalk コンソールを使用した静的ファイルの設定の詳細については、「the section called “静的ファイル”」を参照してください。

デフォルトでは、Python 環境のプロキシサーバーは `static` パスにある `/static` という名前のフォルダにあるすべてのファイルを提供します。たとえば、アプリケーション出典の `logo.png` という名前のフォルダに `static` というファイルが含まれている場合、プロキシサーバーは *`subdomain`*`.elasticbeanstalk.com/static/logo.png` でそれをユーザーに提供します。このセクションで説明しているように、追加のマッピングを設定できます。

環境プロパティ

環境プロパティを使用して、アプリケーションに情報を渡し、環境変数を設定できます。たとえば、`CONNECTION_STRING` という名前の環境プロパティを作成し、そのプロパティで、データベースへの connect にアプリケーションが使用する接続文字列を指定できます。

Elastic Beanstalk 内で実行している Python 環境の内部では、Python の `os.environ` ディクショナリを使用してこれらの値にアクセスできます。詳細については、http://docs.python.org/library/os.html を参照してください。

以下のようなコードを使用して、キーとパラメータにアクセスできます。

```
import os
endpoint = os.environ['API_ENDPOINT']
```

環境プロパティを使用して、フレームワークに情報を渡すこともできます。たとえば、`DJANGO_SETTINGS_MODULE` という名前のプロパティを作成して、そのプロパティで、特定の設定モジュールを使用するように Django を設定できます。環境によって、値は、`development.settings` や `production.settings` などになります。

詳細については「環境変数およびその他のソフトウェア設定」を参照してください。

Python 設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクをパフォーマンスできます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

Python プラットフォームでは、名前空間 (`aws:elasticbeanstalk:environment:proxy`、`aws:elasticbeanstalk:environment:proxy:s` のオプションを定義します。

以下の設定ファイルの例では、`DJANGO_SETTINGS_MODULE` という名前の環境プロパティを作成する設定オプションを指定し、Apache プロキシサーバーを選択して、`statichtml` という名前のディレクトリを `/html` パスにマッピングし `staticimages` という名前のディレクトリを `/images` パスにマッピングする 2 つの静的ファイルオプションを指定して、`aws:elasticbeanstalk:container:python` 名前空間に追加の設定を指定しています。この名前空間には、出典コードで WSGI スクリプトの場所を指定できるオプションと、WSGI で実行するスレッドとプロセスの数を指定できるオプションが含まれています。

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    DJANGO_SETTINGS_MODULE: production.settings
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /html: statichtml
    /images: staticimages
  aws:elasticbeanstalk:container:python:
    WSGIPath: ebdjango.wsgi:application
    NumProcesses: 3
    NumThreads: 20
```

> **メモ**
>
> - Amazon Linux AMI Python プラットフォームバージョン (Amazon Linux 2 より前の) を使用している場合は、`WSGIPath` の値を `ebdjango/wsgi.py` に置き換えます。この例の値は、Amazon Linux AMI プラットフォームバージョンでは support されていない Gunicorn WSGI サーバーで動作します。

- さらに、これらの古いプラットフォームバージョンでは、静的ファイル `–aws:elasticbeanstalk:container:python:staticfiles`の設定に異なる名前空間が使用されます 。これは、スタンダードの静的ファイル名前空間と同じオプション名とセマンティクスを持っています。

設定ファイルでは、お客様の環境のインスタンスでソフトウェアをさらに変更するためのキーもいくつか support されています。この例では、packages キーを使用して、Memcached をインストールし、yum および コンテナ コマンドを使用して、デプロイ時に run command にサーバーを設定します。

```
packages:
  yum:
    libmemcached-devel: '0.31'

container_commands:
  collectstatic:
    command: "django-admin.py collectstatic --noinput"
  01syncdb:
    command: "django-admin.py syncdb --noinput"
    leader_only: true
  02migrate:
    command: "django-admin.py migrate"
    leader_only: true
  03wsgipass:
    command: 'echo "WSGIPassAuthorization On" >> ../wsgi.conf'
  99customize:
    command: "scripts/customize.sh"
```

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

python3 実行可能ファイル

Elastic Beanstalk Python 環境の EC2 インスタンスで使用できるpython3実行可能ファイルのバージョンは、プラットフォームで使用されるのと同じ Python バージョンに必ずしも対応しているとは限りません。たとえば、Python 3.12 AL2023 プラットフォームでは、 は Python 3.9 `/usr/bin/python3`を指します。これは、Python 3.9 が AL2023 のシステム Python であるためです。詳細については、Amazon Linux AL2023 ユーザーガイドの「AL2023 での Python」を参照してください。

プラットフォームで使用される Python バージョンに対応する実行可能ファイルには、バージョン管理された場所 (例: `/usr/bin/python3.12`) またはアプリケーションの仮想環境`bin`ディレクトリ (例:) からアクセスできます`/var/app/venv/staging-LQM1lest/bin/python3`。プラットフォームは、プラットフォームブランチに対応する正しい Python 実行可能ファイルを使用します。

Elastic Beanstalk での Procfile を使用した WSGI サーバーの設定

ソ出典バンドルに `Procfile` を追加して、アプリケーションの WSGI サーバーを指定および設定することができます。`Procfile` でカスタムのスタートコマンドと実行コマンドを指定できます。

`Procfile` を使用すると、設定ファイルを使用して設定した `aws:elasticbeanstalk:container:python` 名前空間オプションが上書きされます。

次の例では、`Procfile` を使用して uWSGI をサーバーとして指定し、これを設定します。

Example [Procfile]

```
web: uwsgi --http :8000 --wsgi-file application.py --master --processes 4 --threads 2
```

次の例では、`Procfile` を使用して、デフォルトの WSGI サーバーである Gunicorn を設定します。

Example [Procfile]

```
web: gunicorn --bind :8000 --workers 3 --threads 2 project.wsgi:application
```

> **メモ**
>
> - Gunicorn 以外の WSGI サーバーを設定する場合は、アプリケーションの依存関係として指定して、環境インスタンスにインストールされるようにしてください。依存関係の指定の詳細については、「the section called “依存関係の指定”」を参照してください。
> - WSGI サーバーのデフォルトポートは 8000 です。`Procfile` コマンドで別のポート番号を指定する場合は、`PORT` 環境プロパティ もこのポート番号に設定します。

Elastic Beanstalk での要件ファイルを使用した依存関係の指定

このトピックでは、アプリケーションを設定して、必要な他の Python パッケージをインストールする方法について説明します。通常の Python アプリケーションには、他のサードパーティー製

Python パッケージに対する依存関係があります。Elastic Beanstalk Python プラットフォームでは、アプリケーションが依存する Python パッケージを指定する方法が複数あります。

pip および requirements.txt を使用します。

Python パッケージをインストールするためのスタンダードツールは pip です。これには、すべての必要なパッケージ (およびバージョン) を 1 つの要件ファイルで指定する特徴があります。要件ファイルの詳細については、pip ドキュメントウェブサイトの「要件ファイル形式」を参照してください。

requirements.txt という名前のファイルを作成し、出典バンドルの最上位ディレクトリに置きます。次は、Django の requirements.txt ファイルの例です。

```
Django==2.2
mysqlclient==2.0.3
```

開発環境で、pip freeze コマンドを使用して要件ファイルを生成できます。

```
~/my-app$ pip freeze > requirements.txt
```

要件ファイルに、実際にアプリケーションによって使用されるパッケージのみが含まれるようにするには、それらのパッケージのみがインストールされている仮想環境を使用します。仮想環境の外では、pip freeze の出力に、オペレーティングシステムに付属のパッケージを含め、開発マシンにインストール済みのすべての pip パッケージが含まれます。

> **Note**
>
> Amazon Linux AMI Python プラットフォームバージョンでは、Elastic Beanstalk は Pipenv または Pipfiles をネイティブにサポートしていません。Pipenv を使用してアプリケーションの依存関係を管理する場合は、次のコマンドを実行して requirements.txt ファイルを生成します。
>
> ```
> ~/my-app$ pipenv lock -r > requirements.txt
> ```
>
> 詳細については、Pipenv ドキュメントの Generating a requirements.txt を参照してください。

Pipenv と **Pipfile** を使用します

Pipenv は、最新の Python パッケージングツールです。これは、パッケージのインストールと依存関係ファイルの作成と管理およびアプリケーションの virtualenv を組み合わせたものです。詳細については、「Pipenv: 人間のための Python 開発ワークフロー」を参照してください。

Pipenv は 2 つのファイルを維持します。

- `Pipfile` – このファイルには、さまざまな種類の依存関係と要件が含まれています。
- `Pipfile.lock` – このファイルには、確定的なビルドを可能にするバージョンスナップショットが含まれています。

これらのファイルを開発環境で作成し、Elastic Beanstalk にデプロイするソースバンドルの最上位ディレクトリに含めることができます。これらの 2 つのファイルの詳細については、「Pipfile と Pipfile.lock の例」を参照してください。

次の例では、Django と Django REST フレームワークをインストールするために Pipenv を使用しています。これらのコマンドは、ファイル `Pipfile` と `Pipfile.lock` を作成します。

```
~/my-app$ pipenv install django
~/my-app$ pipenv install djangorestframework
```

優先順位

このトピックで説明する要件ファイルを複数含めると、Elastic Beanstalk はそのうちの 1 つだけを使用します。次の表に、優先順位を降順で示します。

1. `requirements.txt`
2. `Pipfile.lock`
3. `Pipfile`

> **Note**
>
> 2023 年 3 月 7 日の Amazon Linux 2 プラットフォームリリース以降、これらのファイルを複数指定すると、Elastic Beanstalk はデプロイ中にどの依存関係ファイルが使用されたかを示すコンソールメッセージを発行します。

次のステップは、インスタンスのデプロイ時に Elastic Beanstalk が依存関係をインストールするために従うロジックです。

- `requirements.txt` ファイルがある場合は、コマンド `pip install -r requirements.txt` を使用します。
- 2023 年 3 月 7 日の Amazon Linux 2 プラットフォームリリース以降、`requirements.txt` ファイルはなくても `Pipfile.lock` がある場合は、コマンド `pipenv sync` を使用します。そのリリース前は、`pipenv install --ignore-pipfile` を使用していました。
- `requirements.txt` ファイルと `Pipfile.lock` のいずれも存在しないが、`Pipfile` が存在する場合は、コマンド `pipenv install --skip-lock` を使用します。
- 3 つの要件ファイルが見つからない場合は、アプリケーションの依存関係はインストールされません。

Elastic Beanstalk への Flask アプリケーションのデプロイ

このチュートリアルでは、Flask アプリケーションを生成して AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。Flask は、Python のオープンソースのウェブアプリケーション・フレームワークです。

このチュートリアルでは、以下の作業を行います。

- Flask で Python 仮想環境を設定する
- Flask アプリケーションを作成する
- EB CLI でサイトをデプロイします
- クリーンアップ

前提条件

このチュートリアルでは、基本的な Elastic Beanstalk オペレーションと Elastic Beanstalk コンソールに関する知識があることを前提としています。まだ起動していない場合は、Elastic Beanstalk の使用を開始する方法について説明します。 の指示に従って、最初の Elastic Beanstalk 環境を起動します。

このガイドの手順に従うには、run command のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

このチュートリアルでは、Python 3.11 と対応する Elastic Beanstalk プラットフォームバージョンを使用します。「Elastic Beanstalk 用の Python 開発環境の設定」の手順に従って Python をインストールします

チュートリアルのパートとして Flask フレームワークをインストールします。

また、このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) も使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

Flask で Python 仮想環境を設定する

アプリケーション用のプロジェクトディレクトリと仮想環境を作成し、Flask をインストールします。

プロジェクト環境をセットアップするには

1. プロジェクトディレクトリを作成します。

   ```
   ~$ mkdir eb-flask
   ~$ cd eb-flask
   ```

2. `virt` という名前の仮想環境を作成してアクティブ化します。

   ```
   ~/eb-flask$ virtualenv virt
   ~$ source virt/bin/activate
   (virt) ~/eb-flask$
   ```

   コマンドプロンプトの先頭に `(virt)` と表示され、仮想環境を使用していることが示されます。このチュートリアルの残りの部分では、仮想環境を使用します。

3. `pip install` で Flask をインストールします

   ```
   (virt)~/eb-flask$ pip install flask==3.0.3
   ```

4. `pip freeze` で、インストールされているライブラリを表示します

```
(virt)~/eb-flask$ pip freeze
blinker==1.8.2
click==8.1.7
Flask==3.0.3
importlib_metadata==8.5.0
itsdangerous==2.2.0
Jinja2==3.1.4
MarkupSafe==2.1.5
Werkzeug==3.0.4
zipp==3.20.2
```

このコマンドは、仮想環境にインストールされるすべてのパッケージを一覧します。仮想環境にいるため、EB CLI などのグローバルにインストールされたパッケージは表示されません。

5. `pip freeze` からの出力を、`requirements.txt` という名前のファイルに保存します。

```
(virt)~/eb-flask$ pip freeze > requirements.txt
```

このファイルは、デプロイ中にライブラリをインストールするよう Elastic Beanstalk に指示します。詳細については、「Elastic Beanstalk での要件ファイルを使用した依存関係の指定」を参照してください。

Flask アプリケーションを作成する

次に、Elastic Beanstalk を使用してデプロイするアプリケーションを作成します。ここでは、"Hello World" という RESTful ウェブサービスを作成します。

このディレクトリに、`application.py` という名前と以下の内容で新しいテキスト・ファイルを作成します。

Example **`~/eb-flask/application.py`**

```
from flask import Flask

# print a nice greeting.
def say_hello(username = "World"):
    return '<p>Hello %s!</p>\n' % username

# some bits of text for the page.
```

```
header_text = '''
    <html>\n<head> <title>EB Flask Test</title> </head>\n<body>'''
instructions = '''
    <p><em>Hint</em>: This is a RESTful web service! Append a username
    to the URL (for example: <code>/Thelonious</code>) to say hello to
    someone specific.</p>\n'''
home_link = '<p><a href="/">Back</a></p>\n'
footer_text = '</body>\n</html>'

# EB looks for an 'application' callable by default.
application = Flask(__name__)

# add a rule for the index page.
application.add_url_rule('/', 'index', (lambda: header_text +
    say_hello() + instructions + footer_text))

# add a rule when the page is accessed with a name appended to the site
# URL.
application.add_url_rule('/<username>', 'hello', (lambda username:
    header_text + say_hello(username) + home_link + footer_text))

# run the app.
if __name__ == "__main__":
    # Setting debug to True enables debug output. This line should be
    # removed before deploying a production app.
    application.debug = True
    application.run()
```

この例では、サービスへのアクセスに使用されるパスに基づいて変更されるカスタマイズされた挨拶を出力します。

> **Note**
>
> アプリケーションを実行する前に `application.debug = True` を追加することで、問題が発生した場合にデバッグ出力が有効になります。これは開発の場合に有用な方法ですが、デバッグ出力によってアプリケーションの内部的な側面が明らかになる可能性があるため、実稼働のコードではデバッグステートメントを削除する必要があります。

ファイル名として `application.py` を使用し、呼び出し可能な `application` オブジェクト（この場合は Flask オブジェクト）を提供することで、Elastic Beanstalk がアプリケーションコードを見つけやすくなります。

Python を使用して `application.py` を実行します。

```
(virt) ~/eb-flask$ python application.py
 * Serving Flask app "application" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 313-155-123
```

ウェブ・ブラウザで `http://127.0.0.1:5000/` を開きます。実行中のアプリケーションが表示され、インデックスページが表示されます。

サーバーログをチェックして、リクエストからの出力を確認します。Ctrl+C を押すと、ウェブ・サーバーを停止して仮想環境に戻ることができます。

デバッグ出力が表示される場合は、Elastic Beanstalk 用に設定する前に、エラーを修正し、そのアプリケーションがローカルで実行されることを確認します。

EB CLI でサイトをデプロイします

Elastic Beanstalk でアプリケーションをデプロイするために必要な条件をすべて追加しました。プロジェクトディレクトリは次のようになります。

```
~/eb-flask/
|-- virt
|-- application.py
`-- requirements.txt
```

ただし、`virt` フォルダは、Elastic Beanstalk でアプリケーションを実行するために必要ありません。デプロイすると、Elastic Beanstalk によりサーバーインスタンスに新しい仮想環境が作成され、`requirements.txt` にリストされているライブラリがインストールされます。デプロイ中にアップロードする出典バンドルのサイズを最小化するには、`virt` フォルダを離れるように EB CLI に指示する .ebignore ファイルを追加します。

Example ~/eb-flask/.ebignore

```
virt
```

次に、アプリケーション環境を作成し、設定済みのアプリケーションを Elastic Beanstalk を使用してデプロイします。

環境を作成し、Flask アプリケーションをデプロイするには

1. eb init コマンドで EB CLI リポジトリを初期化します。

   ```
   ~/eb-flask$ eb init -p python-3.11 flask-tutorial --region us-east-2
   Application flask-tutorial has been created.
   ```

   このコマンドは、 という名前の新しいアプリケーションを作成し`flask-tutorial`、最新の Python 3.11 プラットフォームバージョンで環境を作成するようにローカルリポジトリを設定します。
2. （オプション）eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH で connect できるようにします。

   ```
   ~/eb-flask$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

   1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従って新しいキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。
3. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。

   ```
   ~/eb-flask$ eb create flask-env
   ```

環境の作成の所要時間は約 5 分です。以下のリソースが作成されます。

- EC2 インスタンス – 選択したプラットフォームでウェブ・アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。

- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。

Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

これらのリソースはすべて Elastic Beanstalk によって管理されます。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

> **Note**
>
> Elastic Beanstalk が作成する Amazon S3 バケットは、環境間で共有され、環境の終了時に削除されません。詳細については、「Amazon S3 で Elastic Beanstalk を使用する」を参照してください。

環境作成プロセスが完了したら、eb open でウェブサイトを開きます。

```
~/eb-flask$ eb open
```

これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ・ウィンドウが開きます。ローカルで作成してテストしたのと同じ Flask ウェブサイトが表示されるはずです。

アプリケーションが実行されていることを確認できない場合や、エラーメッセージが表示される場合は、エラーの原因を判断する方法のヒントを得るためにデプロイのトラブルシューティングを参照してください。

アプリケーションが実行されていることを実際に確認できた場合は、Elastic Beanstalk での最初の Flask アプリケーションのデプロイが正常に完了しています。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

EB CLI の場合:

```
~/eb-flask$ eb terminate flask-env
```

次のステップ

Flask の詳細については、flask.pocoo.org を参照してください。

別の Python ウェブ・フレームワークを試してみる場合は、「Elastic Beanstalk への Django アプリケーションのデプロイ」をチェックしてください

Elastic Beanstalk への Django アプリケーションのデプロイ

このチュートリアルでは、自動生成されたデフォルトの Django ウェブサイトを、Python を実行している AWS Elastic Beanstalk 環境にデプロイする手順を説明します。このチュートリアルでは、Elastic Beanstalk 環境を使用して、クラウドで Python ウェブアプリケーションをホストする方法を示します。

このチュートリアルでは、以下の作業を行います。

- Python 仮想環境のセットアップと Django のインストール
- Django プロジェクトを作成する
- Elastic Beanstalk 用に Django アプリケーションを設定する
- EB CLI でサイトをデプロイします
- アプリケーションの更新
- クリーンアップ

前提条件

このチュートリアルを実行するには、以下のパッケージを含む、インストールされている Python の一般的な前提条件をすべて満たす必要があります。

- Python 3.7 以降
- `pip`
- `virtualenv`
- `awsebcli`

チュートリアルのパートとして Django フレームワークをインストールします。

> **Note**
> EB CLI を使用して環境を作成するには、サービスロールが必要です。Elastic Beanstalk コンソールで環境を作成することで、サービスロールを作成できます。サービスロールがない場合は、`eb create` を実行すると EB CLI で作成されます。

Python 仮想環境のセットアップと Django のインストール

`virtualenv` で仮想環境を作成し、それを使用して Django とその依存関係をインストールします。仮想環境を使用することで、アプリケーションで必要なパッケージを正確に識別し、アプリケーションを実行する Amazon EC2 インスタンスにそれらの必要なパッケージをインストールすることができます。

以下のステップは、Unix ベースのシステムと Windows で入力する必要があるコマンドを、それぞれ別のタブに示しています。

開発環境を設定するには

1. `eb-virt` という名前の仮想環境を作成します。

   Unix-based systems

   ```
   ~$ virtualenv ~/eb-virt
   ```

Windows

```
C:\> virtualenv %HOMEPATH%\eb-virt
```

2. 仮想環境をアクティブ化します。

Unix-based systems

```
~$ source ~/eb-virt/bin/activate
(eb-virt) ~$
```

Windows

```
C:\>%HOMEPATH%\eb-virt\Scripts\activate
(eb-virt) C:\>
```

コマンドプロンプトの先頭に (`eb-virt`) と表示され、仮想環境を使用していることが示されます。

> **Note**
>
> 残りの手順では、ホームディレクトリ ~$ での Linux のコマンドプロンプトを示しています。Windows では、`C:\Users\`*`USERNAME`*> であり、*`#######`* は Windows ログイン名です。

3. `pip` を使用して Django をインストールします。

```
(eb-virt)~$ pip install django==2.2
```

> **Note**
>
> インストールする Django のバージョンは、アプリケーションのデプロイに選択した Elastic Beanstalk の Python 設定の Python バージョンと互換性がある必要があります。デプロイの詳細については、このトピックの「???」を参照してください。
> 現在の Python プラットフォームバージョンの詳細については、AWS Elastic Beanstalk プラットフォームドキュメントの「Python」を参照してください。

Django のバージョンと Python との互換性については、「What Python version can I use with Django? (Django で使用可能な Python のバージョンを教えてください)」を参照してください。

4. Django がインストールされたことを確認するには、次のように入力します。

```
(eb-virt)~$ pip freeze
Django==2.2
...
```

このコマンドは、仮想環境にインストールされるすべてのパッケージを一覧します。後でこのコマンドの出力を使用して、プロジェクトを Elastic Beanstalk で使用するために設定します。

Django プロジェクトを作成する

仮想環境を使用して、Django プロジェクトを作成してマシンで実行する準備ができました。

> **Note**
>
> このチュートリアルでは、Python に含まれているデータベースエンジン、SQLite を使用します。データベースはプロジェクトファイルでデプロイされます。本番稼働用環境では、Amazon Relational Database Service (Amazon RDS) を使用して、環境を分離することをお勧めします。詳細については、「Amazon RDS DB インスタンスを Python Elastic Beanstalk 環境に追加する」を参照してください。

Django アプリケーションを作成するには

1. 仮想環境をアクティブ化します。

   Unix-based systems

   ```
   ~$ source ~/eb-virt/bin/activate
   (eb-virt) ~$
   ```

   Windows

   ```
   C:\>%HOMEPATH%\eb-virt\Scripts\activate
   ```

```
(eb-virt) C:\>
```

コマンドプロンプトの先頭に (eb-virt) プレフィックスが表示され、仮想環境を使用していることが示されます。

> **Note**
>
> 残りの手順では、ホームディレクトリと Linux のホームディレクトリ ~/ に Linux のコマンドプロンプト ~$ を示しています。Windows では、C:\Users*USERNAME*> であり、*#######* は Windows ログイン名です。

2. `django-admin startproject` コマンドを使用して、`ebdjango` という名前の Django のプロジェクトを作成します。

```
(eb-virt)~$ django-admin startproject ebdjango
```

このコマンドは、次のディレクトリ構造を持つ ebdjango というスタンダードの Django サイトを作成します。

```
~/ebdjango
  |-- ebdjango
  |   |-- __init__.py
  |   |-- settings.py
  |   |-- urls.py
  |   `-- wsgi.py
  `-- manage.py
```

3. `manage.py runserver` で Django サイトをローカルで実行します。

```
(eb-virt) ~$ cd ebdjango
```

```
(eb-virt) ~/ebdjango$ python manage.py runserver
```

4. ウェブ・ブラウザで `http://127.0.0.1:8000/` を開いて、サイトを表示します。
5. サーバーログをチェックして、リクエストからの出力を確認します。ウェブサーバーを停止して仮想環境に戻るには、 Ctrl+C を押します。

```
Django version 2.2, using settings 'ebdjango.settings'
```

```
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
[07/Sep/2018 20:14:09] "GET / HTTP/1.1" 200 16348
Ctrl+C
```

Elastic Beanstalk 用に Django アプリケーションを設定する

Django を使用してローカルシステムに作成したサイトを、Elastic Beanstalk でのデプロイ用に設定します。

デフォルトでは、Elastic Beanstalk は、アプリケーションを開始するために `application.py` というファイルを探します。これは作成した Django プロジェクトには存在しないため、アプリケーションの環境を調整する必要があります。また、アプリケーションのモジュールをロードできるように、環境可変数を設定する必要もあります。

Elastic Beanstalk 用にサイトを設定するには

1. 仮想環境をアクティブ化します。

   Unix-based systems

   ```
   ~/ebdjango$ source ~/eb-virt/bin/activate
   ```

   Windows

   ```
   C:\Users\USERNAME\ebdjango>%HOMEPATH%\eb-virt\Scripts\activate
   ```

2. `pip freeze` を実行して、出力を `requirements.txt` という名前のファイルに保存します。

   ```
   (eb-virt) ~/ebdjango$ pip freeze > requirements.txt
   ```

   Elastic Beanstalk は `requirements.txt` を使用して、アプリケーションを実行する EC2 インスタンスにどのパッケージをインストールするかを判断します。

3. `.ebextensions` という名前のディレクトリを作成します。

   ```
   (eb-virt) ~/ebdjango$ mkdir .ebextensions
   ```

4. `.ebextensions` ディレクトリ内に、次のテキストを持つ `django.config` という名前の 設定ファイル を追加します。

Example ~/ebdjango/.ebextensions/django.config

```
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: ebdjango.wsgi:application
```

この設定 `WSGIPath` は、アプリケーションを起動するのに Elastic Beanstalk が使用する WSGI スクリプトの場所を指定します。

> **Note**
>
> Amazon Linux AMI Python プラットフォームバージョン (Amazon Linux 2 より前の) を使用している場合は、`WSGIPath` の値を `ebdjango/wsgi.py` に置き換えます。この例の値は、Amazon Linux AMI プラットフォームバージョンでは support されていない Gunicorn WSGI サーバーで動作します。

5. `deactivate` コマンドを使用して、仮想環境を非アクティブ化します。

```
(eb-virt) ~/ebdjango$ deactivate
```

パッケージをアプリケーションに追加するか、またはアプリケーションをローカルで実行する必要があるときは、いつでも仮想環境を再アクティブ化します。

EB CLI でサイトをデプロイします

Elastic Beanstalk でアプリケーションをデプロイするために必要な条件をすべて追加しました。プロジェクトディレクトリは次のようになります。

```
~/ebdjango/
|-- .ebextensions
|   `-- django.config
|-- ebdjango
|   |-- __init__.py
|   |-- settings.py
|   |-- urls.py
|   `-- wsgi.py
|-- db.sqlite3
|-- manage.py
```

```
`-- requirements.txt
```

次に、アプリケーション環境を作成し、設定済みのアプリケーションを Elastic Beanstalk を使用してデプロイします。

デプロイの直後に、Django の設定を編集して Elastic Beanstalk からアプリケーションに割り当てられたドメイン名を Django の `ALLOWED_HOSTS` に追加します。次に、アプリケーションを再デプロイします。これは、HTTP `Host` ヘッダー攻撃を防ぐように設計された Django のセキュリティ要件です。詳細については、「Host header validation (ホスト・ヘッダーの検証)」を参照してください。

環境を作成し、Django アプリケーションをデプロイするには

> **Note**
>
> このチュートリアルでは、EB CLI をデプロイメントメカニズムとして使用しますが、Elastic Beanstalk コンソールを使用してプロジェクトの内容を含む .zip ファイルを展開することもできます。

1. eb init コマンドを使用して EB CLI リポジトリを初期化します。

   ```
   ~/ebdjango$ eb init -p python-3.7 django-tutorial
   Application django-tutorial has been created.
   ```

   このコマンドでは、`django-tutorial` という名前のアプリケーションを作成します。また、ローカルリポジトリを設定し、最新の Python 3.7 プラットフォームバージョンで環境を作成します。

2. (オプション) eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH を使用して connect できるようにします。

   ```
   ~/ebdjango$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従ってキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。

3. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。

```
~/ebdjango$ eb create django-env
```

> **Note**
>
> 「service role required」エラーメッセージが表示された場合は、`eb create` をインタラクティブに (環境名を指定せずに) 実行してください。EB CLI によってロールが作成されます。

このコマンドは、`django-env` という名前のロードバランシング Elastic Beanstalk 環境を作成します。環境の作成には約 5 分かかります。Elastic Beanstalk はアプリケーションを実行するのに必要なリソースを作成し、EB CLI がターミナルに中継する情報メッセージを出力します。

4. 環境の作成プロセスが完了したら、eb status を実行して新しい環境のドメイン名を見つけます。

```
~/ebdjango$ eb status
Environment details for: django-env
  Application name: django-tutorial
  ...
  CNAME: eb-django-app-dev.elasticbeanstalk.com
  ...
```

環境のドメイン名は、`CNAME` プロパティの値です。

5. `ebdjango` ディレクトリの `settings.py` ファイルを開きます。`ALLOWED_HOSTS` 設定を見つけ、前のステップで見つけたアプリケーションのドメイン名を設定の値に追加します。この設定がファイルで見つからない場合は、それを新しい行に追加します。

```
...
ALLOWED_HOSTS = ['eb-django-app-dev.elasticbeanstalk.com']
```

6. ファイルを保存し、eb deploy を実行してアプリケーションをデプロイします。eb deploy を実行すると、EB CLI はプロジェクトディレクトリのコンテンツをバンドルアップして、ユーザーの環境にデプロイします。

```
~/ebdjango$ eb deploy
```

> **Note**
> プロジェクトで Git を使用している場合は、「Git での EB CLI の使用」を参照してください。

7. 環境の更新プロセスが完了したら、eb open でウェブサイトを開きます。

```
~/ebdjango$ eb open
```

これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ・ウィンドウが開きます。ローカルで作成してテストしたのと同じ Django ウェブサイトが表示されます。

アプリケーションが実行されていることを確認できない場合や、エラーメッセージが表示される場合は、エラーの原因を判断する方法のヒントを得るためにデプロイのトラブルシューティングを参照してください。

アプリケーションが実行されていることを実際に確認できた場合は、Elastic Beanstalk での最初の Django アプリケーションのデプロイが正常に完了しています。

アプリケーションの更新

Elastic Beanstalk で実行中のアプリケーションまたはその設定を更新して再デプロイすることができます。この場合、インスタンスを更新して新しいアプリケーションバージョンを開始する作業は、Elastic Beanstalk が実行します。

この例では、Django の管理者コンソールを有効にして、他のいくつかの項目を設定します。

サイト設定を変更する

デフォルトでは、Django ウェブサイトは UTC タイムゾーンを使用して時間を表示します。`settings.py` でタイムゾーンを指定して、これを変更することができます。

サイトのタイムゾーンを変更するには

1. `settings.py` の `TIME_ZONE` 設定を変更します

   Example ~/ebdjango/ebdjango/settings.py

   ```
   ...
   # Internationalization
   LANGUAGE_CODE = 'en-us'
   TIME_ZONE = 'US/Pacific'
   USE_I18N = True
   USE_L10N = True
   USE_TZ = True
   ```

   タイムゾーンのリストについては、このページを参照してください。

2. アプリケーションを Elastic Beanstalk 環境にデプロイします。

   ```
   ~/ebdjango/$ eb deploy
   ```

サイト管理者を作成する

Django アプリケーション用のサイト管理者を作成すると、ウェブサイトから管理者コンソールに直接アクセスできます。管理者のログインの詳細は、Django が生成したデフォルトプロジェクトに含まれるローカルデータベース・イメージに安全に保存されます。

サイト管理者を作成するには

1. Django アプリケーションのローカルデータベースを初期化します。

   ```
   (eb-virt) ~/ebdjango$ python manage.py migrate
   Operations to perform:
     Apply all migrations: admin, auth, contenttypes, sessions
   Running migrations:
     Applying contenttypes.0001_initial... OK
     Applying auth.0001_initial... OK
     Applying admin.0001_initial... OK
     Applying admin.0002_logentry_remove_auto_add... OK
     Applying admin.0003_logentry_add_action_flag_choices... OK
     Applying contenttypes.0002_remove_content_type_name... OK
     Applying auth.0002_alter_permission_name_max_length... OK
   ```

```
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
```

2. `manage.py createsuperuser` を実行して、管理者を作成します。

```
(eb-virt) ~/ebdjango$ python manage.py createsuperuser
Username: admin
Email address: me@mydomain.com
Password: ********
Password (again): ********
Superuser created successfully.
```

3. 静的ファイルの保存する場所を Django に渡すには、`STATIC_ROOT` で `settings.py` を定義します。

   Example ~/ebdjango/ebdjango/settings.py

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.2/howto/static-files/
STATIC_URL = '/static/'
STATIC_ROOT = 'static'
```

4. `manage.py collectstatic` を実行して、`static` ディレクトリに管理者サイトの静的アセット (JavaScript、CSS、イメージ) を追加します。

```
(eb-virt) ~/ebdjango$ python manage.py collectstatic
119 static files copied to ~/ebdjango/static
```

5. アプリケーションをデプロイします。

```
~/ebdjango$ eb deploy
```

6. ブラウザでサイトを開き、次のようにサイトの URL に `/admin/` を付加することで、管理者コンソールが表示されます。

```
http://djang-env.p33kq46sfh.us-west-2.elasticbeanstalk.com/admin/
```

7. ステップ 2 で設定したユーザーネームとパスワードでログ・インします。

ローカルでの更新/テストと同様の手順を使用し、続けて eb deploy を実行することができます。実稼働サーバーの更新は Elastic Beanstalk が処理するため、ユーザーはサーバー管理ではなくアプリケーション開発に集中できます。

データベース移行設定ファイルを追加する

サイトが更新されたときに実行される .ebextensions スクリプトに、コマンドを追加することができます。これにより、データベース・マイグレーションを自動的に生成できます。

アプリケーションがデプロイされたときに移行ステップを追加するには

1. 以下の内容を使用して、db-migrate.config と言う名前の設定ファイルを追加します。

   Example ~/ebdjango/.ebextensions/db-migrate.config

   ```
   container_commands:
     01_migrate:
       command: "source /var/app/venv/*/bin/activate && python3 manage.py migrate"
       leader_only: true
   option_settings:
     aws:elasticbeanstalk:application:environment:
       DJANGO_SETTINGS_MODULE: ebdjango.settings
   ```

   この設定ファイルは、サーバーの仮想環境をアクティブ化し、アプリケーションがスタートする前に、デプロイプロセスの間に manage.py migrate コマンドを実行します。アプリケーションがスタートする前に実行されるため、DJANGO_SETTINGS_MODULE 環境可変数を明確に設定する必要もあります (通常は、スタートアップ中に wsgi.py がこれを行います)。コマンドで leader_only: true を指定することで、複数のインスタンスにデプロイする場合に 1 回だけ実行するようにできます。

2. アプリケーションをデプロイします。

```
~/ebdjango$ eb deploy
```

クリーンアップ

開発セッション間でインスタンス時間やその他の AWS リソースを節約するには、 を使用して Elastic Beanstalk 環境を終了しますeb terminate。

```
~/ebdjango$ eb terminate django-env
```

このコマンドは、環境とその環境内で実行されるすべての AWS リソースを終了します。これによってアプリケーションが削除されることはありません。したがって eb create を再び実行することで、いつでも同じ設定でさらに環境を作成することができます。

サンプルアプリケーションが必要なくなった場合は、プロジェクトフォルダーと仮想環境を削除することもできます。

```
~$ rm -rf ~/eb-virt
~$ rm -rf ~/ebdjango
```

次のステップ

詳細なチュートリアルを含む Django の情報については、公式ドキュメントを参照してください。

別の Python ウェブフレームワークを試す場合は、「Elastic Beanstalk への Flask アプリケーションのデプロイ」をチェックしてください。

Amazon RDS DB インスタンスを Python Elastic Beanstalk 環境に追加する

このトピックでは、Elastic Beanstalk コンソールを使用して Amazon RDS を作成する手順について説明します。Amazon Relational Database Service (Amazon RDS) DB インスタンスを使用して、アプリケーションによって収集および変更されたデータを保存することができます。データベースを環境に結合して Elastic Beanstalk で管理することも、分離したものとして作成して別のサービスで外部的に管理することもできます。これらの手順では、データベースは環境に結合され、Elastic Beanstalk によって管理されます。Amazon RDS と Elastic Beanstalk の統合の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

セクション

- 環境に DB インスタンスを追加
- ドライバのダウンロード
- データベースへの接続

環境に DB インスタンスを追加

お客様の環境に DB インスタンスを追加するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. DB エンジンを選択して、ユーザー名とパスワードを入力します。
6. ページの最下部で [適用] を選択し変更を保存します。

DB インスタンスの追加には約 10 分かかります。環境の更新が完了すると、DB インスタンスのホスト名とその他の接続情報は以下の環境プロパティを通じてアプリケーションに使用できるようになります。

プロパティ名	説明	プロパティ値
`RDS_HOSTNAME`	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
`RDS_PORT`	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。

プロパティ名	説明	プロパティ値
RDS_DB_NAME	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
RDS_USERNAME	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。
RDS_PASSWORD	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

Elastic Beanstalk 環境と結合したデータベースインスタンスの設定の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

ドライバのダウンロード

プロジェクトの要件ファイルにデータベース・ドライバを追加します。

Example requirements.txt – Django with MySQL

```
Django==2.2
mysqlclient==2.0.3
```

Python の共通ドライバパッケージ

- MySQL – `mysqlclient`
- PostgreSQL – `psycopg2`
- Oracle – `cx_Oracle`
- SQL Server – `adodbapi`

詳細については、Python DatabaseInterfaces と Django 2.2 が support するデータベースを参照してください。

データベースへの接続

Elastic Beanstalk は、環境プロパティでアタッチされた DB インスタンスの接続情報を提供します。`os.environ['VARIABLE']` を使用してプロパティを読み取り、データベース接続を設定します。

Example Django の設定ファイル – DATABASES ディクショナリ

```
import os

if 'RDS_HOSTNAME' in os.environ:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': os.environ['RDS_DB_NAME'],
            'USER': os.environ['RDS_USERNAME'],
            'PASSWORD': os.environ['RDS_PASSWORD'],
            'HOST': os.environ['RDS_HOSTNAME'],
            'PORT': os.environ['RDS_PORT'],
        }
    }
```

Python のツールとリソース

Python アプリケーションのデプロイに関しては他にも参照情報があります。

リソース	説明
AWS SDK for Python (Boto3) GitHub の	GitHub から入手した Boto3 をインストールします。
AWS SDK for Python (Boto3) ホームページ	AWS SDK for Python (Boto3) ホームページ。
Python デベロッパーセンター	サンプルコード、ドキュメント、ツール、追加リソースを 1 か所で入手できる場所です。

Elastic Beanstalk での Ruby アプリケーションのデプロイ

この章では、Ruby ウェブアプリケーションを設定してデプロイする手順について説明します AWS Elastic Beanstalk。Elastic Beanstalk を使用すると、Amazon Web Services を使用して簡単に Ruby ウェブアプリケーションのデプロイ、管理、スケーリングができます。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) または Elastic Beanstalk コンソールを使用すると、わずか数分でアプリケーションをデプロイできます。Elastic Beanstalk アプリケーションをデプロイした後も、EB CLI を使用してアプリケーションと環境を管理することも、Elastic Beanstalk コンソール AWS CLIまたは APIs を使用することもできます。

この章では、EB CLI を使用して Elastic Beanstalk にサンプルアプリケーションをデプロイした後、Rails および Sinatra ウェブアプリケーションフレームワークを使用するようにアプリケーションを更新するステップバイステップの手順も説明します。

この章のトピックは、Elastic Beanstalk 環境についてある程度の知識がある方向けであることを前提としています。Elastic Beanstalk を使用したことがない場合は、入門ガイドチュートリアルで基本知識を得てください。

トピック

- Elastic Beanstalk 用の Ruby 開発環境の設定
- Elastic Beanstalk Ruby プラットフォームを使用する
- Elastic Beanstalk への Rails アプリケーションのデプロイ
- Elastic Beanstalk への Sinatra アプリケーションのデプロイ
- Amazon RDS DB インスタンスを Ruby Elastic Beanstalk 環境に追加する

Elastic Beanstalk 用の Ruby 開発環境の設定

この章では、Ruby 開発環境を設定し、アプリケーションを AWS Elastic Beanstalkにデプロイする前にローカルでテストする手順について説明します。また、便利なツールのインストール手順を提供するウェブサイトも参照します。

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

セクション

- Ruby のインストール
- AWS SDK for Ruby のインストール
- IDE またはテキストエディタをインストールする

Ruby のインストール

C コンパイラをお持ちでない場合は、GCC をインストールしてください。Ubuntu では `apt` を使用します。

```
~$ sudo apt install gcc
```

Amazon Linux では、`yum` を使用します。

```
~$ sudo yum install gcc
```

Ruby 言語のインストールを管理するための RVM をマシンにインストールします。rvm.io のコマンドを使用してプロジェクトキーを取得し、インストールスクリプトを実行します。

```
~$ gpg2 --recv-keys key1 key2
~$ curl -sSL https://get.rvm.io | bash -s stable
```

このスクリプトは、ユーザディレクトリの `.rvm` という名前のフォルダに RVM をインストールし、新しい端末を開くたびにセットアップスクリプトをロードするようにシェルプロファイルを変更します。スタートするには、スクリプトをマニュアルでロードします。

```
~$ source ~/.rvm/scripts/rvm
```

最新バージョンを取得するには、`rvm get head` を使用します。

```
~$ rvm get head
```

Ruby の使用可能なバージョンを表示します。

```
~$ rvm list known
```

AWS Elastic Beanstalk プラットフォームドキュメントの「Ruby」で、Elastic Beanstalk プラットフォームで利用可能な最新バージョンの Ruby を確認してください。そのバージョンをインストールします。

```
~$ rvm install 3.2
```

Ruby のインストールをテストします。

```
~$ ruby --version
```

AWS SDK for Ruby のインストール

アプリケーション内から AWS リソースを管理する必要がある場合は、 をインストールします AWS SDK for Ruby。例えば、SDK for Ruby では、Amazon DynamoDB (DynamoDB) を使用して、リレーショナルデータベースを作成せずに、ユーザーとセッション情報を保存することができます。

gem コマンドを使用して、SDK for Ruby とその依存関係をインストールします。

```
$ gem install aws-sdk
```

詳細とインストール手順については、AWS SDK for Ruby のホームページを参照してください。

IDE またはテキストエディタをインストールする

統合された開発環境（IDEs）は、アプリケーション開発を容易にする幅広い特徴を提供します。Ruby 開発用の IDE を使用していない場合は、Aptana と RubyMine を試してどちらが使いやすいかを確認してください。

- Install Aptana
- RubyMine

> **Note**
>
> IDE では、出典コントロールにコミットする必要がないファイルがプロジェクトフォルダに追加される場合があります。ソースコントロールにこれらのファイルがコミットされないようにするには、`.gitignore` または同等のソースコントロールツールを使用します。

IDE の特徴のすべては必要なく、単純にコーディングを開始する場合は、Sublime Text のインストールを検討します。

Elastic Beanstalk Ruby プラットフォームを使用する

このトピックでは、Elastic Beanstalk で Ruby アプリケーションを設定、ビルド、実行する方法について説明します。

AWS Elastic Beanstalk は、Ruby プログラミング言語のさまざまなバージョンの多くのプラットフォームブランチをサポートしています。完全なリストについては、「AWS Elastic Beanstalk プラットフォーム」ドキュメントの「Ruby」を参照してください。

Ruby ウェブアプリケーションは、Puma アプリケーションサーバーの NGINX プロキシサーバーの背後で実行できます。RubyGems を使用すると、デプロイの間にパッケージをインストールするためにソースバンドルに `Gemfile` を含めることができます。

アプリケーションサーバーの設定

Elastic Beanstalk は、環境を作成するときに選択した Ruby プラットフォームブランチに基づいて Puma アプリケーションサーバーをインストールします。Ruby プラットフォームのバージョンに付属するコンポーネントの詳細については、AWS Elastic Beanstalk プラットフォームガイドの「サポートされているプラットフォーム」を参照してください。

独自に用意された Puma サーバーでアプリケーションを設定できます。これにより、Ruby プラットフォームブランチにあらかじめインストールされているバージョン以外のバージョンの Puma を使用するオプションが提供されます。Passenger などの別のアプリケーションサーバーを使用するようにアプリケーションを設定することもできます。そのためには、デプロイで `Gemfile` をインクルードしてカスタマイズする必要があります。また、アプリケーションサーバーを起動するように `Procfile` を設定する必要もあります。詳細については、「Procfile でアプリケーションプロセスを設定する」を参照してください。

その他の設定オプション

Elastic Beanstalk には、Elastic Beanstalk 環境内の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスで実行されるソフトウェアをカスタマイズするために使用できる設定オプションが用意されています。アプリケーションに必要な環境変数を設定し、Amazon S3 に対してログのローテーションを有効にしたら、アプリケーションの出典で静的ファイルが含まれるフォルダを、プロキシサーバーによって提供されるパスにマッピングできます。プラットフォームでは、Rails および Rack の検出と使用を容易にすることに関連していくつかの一般的な環境可変数を事前に定義します。

設定オプションは実行中の環境の設定を変更するために Elastic Beanstalk コンソールで利用できます。環境を終了したときにその設定が失われないようにするため、保存済み設定を使用して設定を保存し、それを後で他の環境に適用することができます。

ソースコードの設定を保存する場合、設定ファイルを含めることができます。設定ファイルの設定は、環境を作成するたびに、またはアプリケーションをデプロイするたびに適用されます。設定ファイルを使用して、デプロイの間にパッケージをインストールしたり、スクリプトを実行したり、他のインスタンスのカスタマイズオペレーションを実行することもできます。

Elastic Beanstalk コンソールで適用される設定は、設定ファイルに同じ設定があれば、それらの設定を上書きします。これにより、設定ファイルでデフォルト設定を定義し、コンソールでそのデフォルト設定を環境固有の設定で上書きできます。設定の優先順位の詳細と設定の他の変更方法については、「設定オプション」を参照してください。

Elastic Beanstalk Linux ベースのプラットフォームを拡張するさまざまな方法の詳細については、「the section called “Linux プラットフォームの拡張”」を参照してください。

Ruby 環境の設定

Elastic Beanstalk コンソールを使用して、Amazon S3 へのログローテーションを有効にし、アプリケーションが環境から読むことができる変数を設定することができます。

お客様の環境のソフトウェア構成設定にアクセスするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。

ログオプション

[ログオプション] セクションには、2 つの設定があります。

- [Instance profile] – アプリケーションに関連付けられた Amazon S3 バケットへのアクセス許可が付与されているインスタンスプロファイルを指定します。
- [Enable log file rotation to Amazon S3] (Amazon S3 へのログファイルのローテーションの有効化) - アプリケーションの Amazon EC2 インスタンスのログファイルを、アプリケーションに関連付けられている Amazon S3 バケットにコピーするかどうかを指定します。

静的ファイル

パフォーマンスを向上させるために、[Static files] (静的ファイル) セクションを使用して、ウェブアプリケーション内のディレクトリセットから静的ファイル (HTML、イメージなど) を配信するようにプロキシサーバーを設定することができます。ディレクトリごとに、仮想パスをディレクトリマッピングに設定します。プロキシサーバーは、指定されたパスのファイルに対するリクエストを受け取ると、アプリケーションにリクエストをルーティングする代わりにファイルを直接　　処理します。

設定ファイルまたは、Elastic Beanstalk コンソールを使用した静的ファイルの設定の詳細については、「the section called “静的ファイル”」を参照してください。

デフォルトでは、Ruby 環境のプロキシサーバーは次のように静的ファイルを供するように設定されています。

- `public`フォルダ内のファイルは、`/public`パスとドメインルート (/パス)から供されます。
- `public/assets`サブフォルダ内のファイルは、`/assets`パスです。

以下の例では、デフォルトの設定作業のしくみを示しています。

- たとえば、アプリケーションの出典に `public` という名前のフォルダの `logo.png` というファイルが含まれている場合、プロキシサーバーは *`subdomain`*`.elasticbeanstalk.com/public/logo.png`と*`subdomain`*`.elasticbeanstalk.com/logo.png` からそれをユーザーに提供します。
- たとえば、アプリケーションの出典に `public` のフォルダ内の`assets` という名前のフォルダに `logo.png` というファイルが含まれている場合、プロキシサーバーは *`subdomain`*`.elasticbeanstalk.com/assets/logo.png` でそれをユーザーに提供します。

静的ファイルに追加のマッピングを設定することができます。詳細については、このトピックで後述する「Ruby 設定の名前空間」を参照してください。

> **Note**
>
> より前のプラットフォームバージョン用Ruby 2.7 AL2 バージョン 3.3.7のデフォルトの Elastic Beanstalk nginx プロキシサーバー設定は、ドメインルートからの静的ファイルの提供を support していません (`subdomain.elasticbeanstalk.com/`)。このプラットフォーム版は2021年10月21日にリリースされました。詳細については、「」を参照してください。新しいプラットフォームのバージョン-RubyのAWS Elastic Beanstalk リリースノート。

環境プロパティ

環境プロパティ セクションでは、アプリケーションを実行している Amazon EC2 インスタンスの環境設定を指定できます。環境プロパティは、キーバリューのペアでアプリケーションに渡されます。

Ruby プラットフォームでは、環境設定用に次のプロパティを定義します。

- [BUNDLE_WITHOUT] – Gemfile から依存関係をインストールするときに無視するグループの、コロン区切りのリストです。
- BUNDLER_DEPLOYMENT_MODE – `true` (デフォルト) に設定すると、Bundler を使用してデプロイ・モードで依存関係がインストールされます。開発モードで `bundle install` を実行するには、 `false` に設定します。

  > **Note**
  >
  > この環境プロパティは、Amazon Linux AMI Ruby プラットフォームブランチ (Amazon Linux 2 以前) では定義されていません。

- [RAILS_SKIP_ASSET_COMPILATION] – デプロイの間に、`rake assets:precompile` の実行をスキップするには `true` に設定します。
- [RAILS_SKIP_MIGRATIONS] – デプロイの間に、`rake db:migrate` の実行をスキップするには `true` に設定します。
- [RACK_ENV] – Rack の環境ステージを指定します。例えば、`development`、`production`、または `test` です。

Elastic Beanstalk で実行される Ruby 環境内では、ENV オブジェクトを使用して環境変数にアクセスできます。たとえば、次のコードを使用して可変数に API_ENDPOINT という名前のプロパティを読み取ることができます。

```
endpoint = ENV['API_ENDPOINT']
```

詳細については、「環境変数およびその他のソフトウェア設定」を参照してください。

Ruby 設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクをパフォーマンスできます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

aws:elasticbeanstalk:environment:proxy:staticfiles 名前空間を使用して、静的ファイルを配信する環境プロキシを設定できます。アプリケーションディレクトリへの仮想パスのマッピングを定義します。

Ruby プラットフォームでは、プラットフォーム固有の名前空間を定義しません。代わりに、一般的な Rails および Rack オプションの環境プロパティを定義します。

次の設定ファイルは、staticimages という名前のディレクトリをパス /images にマップして、プラットフォーム定義の各環境プロパティを設定し、LOGGING という名前の追加の環境プロパティを設定する静的ファイルオプションを指定します。

Example .ebextensions/ruby-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /images: staticimages
  aws:elasticbeanstalk:application:environment:
    BUNDLE_WITHOUT: test
    BUNDLER_DEPLOYMENT_MODE: true
    RACK_ENV: development
    RAILS_SKIP_ASSET_COMPILATION: true
    RAILS_SKIP_MIGRATIONS: true
    LOGGING: debug
```

> **Note**
>
> `BUNDLER_DEPLOYMENT_MODE` 環境プロパティと `aws:elasticbeanstalk:environment:proxy:staticfiles` 名前空間は、Amazon Linux AMI Ruby プラットフォームブランチ (Amazon Linux 2 以前) では定義されていません。

Elastic Beanstalk には、環境をカスタマイズするための多数の設定オプションが用意されています。設定ファイルに加えて、コンソール、保存された設定、EB CLI、または を使用して、設定オプションを指定することもできます AWS CLI詳細については「設定オプション」を参照してください。

Elastic Beanstalk での Gemfile を使用したパッケージのインストール

RubyGems を使用してアプリケーションに必要なパッケージをインストールするには、プロジェクトソースのルートに `Gemfile` ファイルを含めます。

Example Gemfile

```
source "https://rubygems.org"
gem 'sinatra'
gem 'json'
gem 'rack-parser'
```

`Gemfile` ファイルがある場合、Elastic Beanstalk は `bundle install` を実行して依存関係をインストールします。詳細については、Bundler.io のウェブサイトにある「Gemfile」と「Bundle」の各ページを参照してください。

> **Note**
>
> Ruby プラットフォームにあらかじめインストールされているデフォルトの他に、別のバージョンの Puma を使用できます。そのためには、バージョンを指定する `Gemfile` をエントリに含めてください。カスタマイズされた `Gemfile` を使用して、Passenger などの別のアプリケーションサーバーを指定することもできます。
> どちらの場合でも、アプリケーションサーバーを起動するように `Procfile` を設定する必要があります。
> 詳細については、「Procfile でアプリケーションプロセスを設定する」を参照してください。

Elastic Beanstalk の Procfile でアプリケーションプロセスを設定する

Ruby アプリケーションを起動するコマンドを指定するには、出典バンドルのルートに `Procfile` というファイルを含めます。

> **Note**
>
> Elastic Beanstalk は、Amazon Linux AMI Ruby プラットフォームブランチ (Amazon Linux 2 以前) でこの特徴をsupport していません。with Puma または with Passenger を含む名前を持つプラットフォームブランチは、Ruby のバージョンにかかわらず `Procfile` の前に付けられ、Amazon Linux 2 の特徴を support しません。

`Procfile` 書き込みと使用の詳細については、「ビルドファイルと Procfile」を参照してください。

を指定しない場合`Procfile`、Elastic Beanstalk はデフォルトの を生成します`Procfile`。に Puma `Gemfile`が含まれている場合、Elastic Beanstalk は指定されたバージョンの Puma を使用すると仮定し、次のデフォルト を生成します`Procfile`。

```
web: bundle exec puma -C /opt/elasticbeanstalk/config/private/pumaconf.rb
```

に Puma が含まれ`Gemfile`ていない場合、Elastic Beanstalk はプリインストールされた Puma アプリケーションサーバーを使用していると見なし、次のデフォルト を生成します`Procfile`。Amazon Linux 2 Ruby プラットフォームブランチ`Procfile`では、 を指定しない場合、Elastic Beanstalk は常に次のデフォルトを生成します`Procfile`。

```
web: puma -C /opt/elasticbeanstalk/config/private/pumaconf.rb
```

> **Note**
>
> 2024 年 10 月 10 日、最後の Ruby Amazon Linux 2 プラットフォームブランチは廃止されました。現在サポートされているすべての Ruby プラットフォームブランチは、Amazon Linux 2023 に基づいています。移行の詳細については、「」を参照してくださいAmazon Linux 2 から Amazon Linux 2023 への移行。

Passenger アプリケーションサーバーを使用する場合は、次のサンプルファイルを使用して、Passenger をインストールして使用するよう Ruby 環境を設定します。

1. このサンプルファイルを使用して Passenger をインストールします。

   Example Gemfile

   ```
   source 'https://rubygems.org'
   gem 'passenger'
   ```

2. このサンプルファイルを使用して Passenger を起動するよう Elastic Beanstalk に指示します。

   Example [Procfile]

   ```
   web: bundle exec passenger start /var/app/current --socket /var/run/puma/my_app.sock
   ```

> **Note**
>
> Passenger を使用するために nginx プロキシサーバーの設定に変更を加える必要はありません。他のアプリケーションサーバーを使用するには、nginx 設定をカスタマイズして、リクエストをアプリケーションに適切に転送する必要があります。

Elastic Beanstalk への Rails アプリケーションのデプロイ

Rails は Ruby 用のオープンソースのモデルビュー・コントローラー (MVC) フレームワークです。このチュートリアルでは、Rails アプリケーションを生成して AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

セクション

- 前提条件
- Elastic Beanstalk の基本的な知識
- Elastic Beanstalk 環境の起動
- rails をインストールしてウェブサイトを生成します
- rails 設定の構成
- アプリケーションをデプロイします
- クリーンアップ
- 次のステップ

前提条件

Elastic Beanstalk の基本的な知識

このチュートリアルでは、基本的な Elastic Beanstalk オペレーションと Elastic Beanstalk コンソールに関する知識があることを前提としています。まだ起動していない場合は、Elastic Beanstalk の使用を開始する方法について説明します。 の指示に従って、最初の Elastic Beanstalk 環境を起動します。

コマンドライン

このガイドの手順に従うには、run command のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

Rails の依存関係

Rails フレームワーク 6.1.4.1 には次の依存関係があります。それらのすべてがインストールされていることを確認します。

- Ruby 2.5.0 以降 – インストール手順については、「Elastic Beanstalk 用の Ruby 開発環境の設定」を参照してください。

  このチュートリアルでは、Ruby 3.0.2 と対応する Elastic Beanstalk プラットフォームバージョンを使用します。
- Node.js – インストール手順については、「Installing Node.js via package manager」を参照してください。
- Yarn – インストールの手順については、Yarn のウェブサイトにある「Installation (インストール)」を参照してください。

Elastic Beanstalk 環境の起動

Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境を作成します。[Ruby] プラットフォームを選択し、デフォルト設定およびサンプルコードを受け入れます。

環境を起動するには (コンソール)

1. 事前に設定されたリンク (console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced) を使用して、Elastic Beanstalk コンソールを開きます。
2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。
3. アプリケーションコード として、サンプルアプリケーション を選択します。
4. 確認と起動 を選択します。
5. 使用できるオプションを確認します。使用する有効なオプションを選択し、準備ができたら [アプリケーションの作成] を選択します。

環境の作成の所要時間は約 5 分です。以下のリソースが作成されます。

- EC2 インスタンス – 選択したプラットフォームでウェブ・ アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・ アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの

HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。

- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

これらのリソースはすべて Elastic Beanstalk によって管理されます。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

> **Note**
>
> Elastic Beanstalk が作成する Amazon S3 バケットは、環境間で共有され、環境の終了時に削除されません。詳細については、「Amazon S3 で Elastic Beanstalk を使用する」を参照してください。

rails をインストールしてウェブサイトを生成します

gem コマンドを使用して、Rails とその依存関係をインストールします。

```
~$ gem install rails
Fetching: concurrent-ruby-1.1.9.gem
Successfully installed concurrent-ruby-1.1.9
Fetching: rack-2.2.3.gem
Successfully installed rack-2.2.3
...
```

Rails のインストールをテストします。

```
~$ rails --version
Rails 6.1.4.1
```

アプリケーションの名前と共に `rails new` を使用して、新しい Rails プロジェクトを作成します。

```
~$ rails new ~/eb-rails
```

Rails によって、指定した名前を持つディレクトリが作成され、サンプルプログラムをローカルで実行する際に必要となるすべてのファイルが生成されます。その後で Bundler が実行され、プロジェクトの Gemfile で定義されている依存関係（Gem）がすべてインストールされます。

> **Note**
>
> このプロセスにより、プロジェクトの最新の Puma バージョンがインストールされます。このバージョンは、Elastic Beanstalk がお使いの環境の Ruby プラットフォームのバージョンに提供しているものとは異なる場合があります。Elastic Beanstalk で提供されている Puma のバージョンについては、AWS Elastic Beanstalk プラットフォームガイドの「Ruby プラットフォーム履歴」を参照してください。最新の Puma バージョンの詳細については、Puma.io のウェブサイトを参照してください。2 つの Puma バージョン間に矛盾があるときは、次のいずれかのオプションを使用します。
>
> - 以前の `rails new` コマンドによりインストールした Puma バージョンを使用する。この場合は、プラットフォームに `Procfile` を追加して、自分の Puma サーバーバージョンを使用します。詳細については、「Elastic Beanstalk の Procfile でアプリケーションプロセスを設定する」を参照してください。

- Puma のバージョンを更新して、お使いの環境の Ruby プラットフォームバージョンにプリインストールされているバージョンに一致させる。これを行うには、プロジェクトソースディレクトリのルートにある Gemfile の Puma バージョンを変更します。次に、`bundle update` を実行します。詳細については、Bundler.io のウェブサイトにある「bundle update (バンドルの更新)」ページを参照してください。

デフォルトのプロジェクトをローカルで実行して、Rails インストールをテストします。

```
~$ cd eb-rails
~/eb-rails$ rails server
=> Booting Puma
=> Rails 6.1.4.1 application starting in development
=> Run `bin/rails server --help` for more startup options
Puma starting in single mode...
* Puma version: 5.5.2 (ruby 3.0.2-p107) ("Zawgyi")
*  Min threads: 5
*  Max threads: 5
*  Environment: development
*          PID: 77857
* Listening on http://127.0.0.1:3000
* Listening on http://[::1]:3000
Use Ctrl-C to stop
...
```

デフォルトのプロジェクトのアクションを確認するには、ウェブ・ブラウザで `http://localhost:3000` を開きます。

このページは、開発モードでのみ表示されます。アプリケーションのフロントページにいくつかのコンテンツを追加して、Elastic Beanstalk への実稼働デプロイメントをサポートします。`rails generate` を使用して、コントローラー、ルート、および先頭ページのビューを作成します。

```
~/eb-rails$ rails generate controller WelcomePage welcome
      create  app/controllers/welcome_page_controller.rb
       route  get 'welcome_page/welcome'
      invoke  erb
      create    app/views/welcome_page
      create    app/views/welcome_page/welcome.html.erb
      invoke  test_unit
      create    test/controllers/welcome_page_controller_test.rb
```

```
      invoke  helper
      create    app/helpers/welcome_page_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/welcome_page.coffee
      invoke    scss
      create      app/assets/stylesheets/welcome_page.scss.
```

これにより、`/welcome_page/welcome` のページにアクセスするのに必要なすべてが提供されます。ただし、変更を発行する前に、ビューのコンテンツを変更し、このページがサイトの最上位レベルに表示されるようにルートを追加します。

テキストエディターを使用して `app/views/welcome_page/welcome.html.erb` の内容を編集します。この例では、`cat` を使用して既存のファイルのコンテンツを上書きします。

Example app/views/welcome_page/welcome.html.erb

```
<h1>Welcome!</h1>
<p>This is the front page of my first Rails application on Elastic Beanstalk.</p>
```

最後に、次のルートを `config/routes.rb` に追加します。

Example config/routes.rb

```
Rails.application.routes.draw do
  get 'welcome_page/welcome'
  root 'welcome_page#welcome'
```

これにより、Rails は、リクエストをウェブサイトのルートに送り、先頭ページのコントローラーの welcome メソッドにルーティングできるようになります。このメソッドでは、先頭ページのビュー（`welcome.html.erb`）のコンテンツを表示します。

Elastic Beanstalk が Ruby プラットフォームにアプリケーションを正常にデプロイするには、更新する必要があります。`Gemfile.lock`。`Gemfile.lock` のいくつかの依存関係は、プラットフォーム固有かもしれません。したがって、追加する必要があります**`platform ruby`**に`Gemfile.lock`必要な依存関係がすべてデプロイと共にインストールされるようにします。

Example

```
~/eb-rails$ bundle lock --add-platform ruby
```

```
Fetching gem metadata from https://rubygems.org/............
Resolving dependencies...
Writing lockfile to /Users/janedoe/EBDPT/RubyApps/eb-rails-doc-app/Gemfile.lock
```

rails 設定の構成

Elastic Beanstalk コンソールを使用して、環境プロパティで Rails を設定します。最大 256 文字の英数字の文字列の SECRET_KEY_BASE 環境プロパティを設定します。

Rails はこのプロパティを使用してキーを作成します。したがって、これは秘密に保ち、プレーンテキストで出典管理に保存しないでください。その代わり、環境プロパティを介して環境の Rails コードに提供します。

Elastic Beanstalk コンソールで環境変数を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. Runtime 環境変数まで下にスクロールします。
6. 環境変数の追加 を選択します。
7. ソース プレーンテキストを選択します。

> **Note**
>
> ドロップダウンの Secrets Manager と SSM パラメータストアの値は、認証情報や API キーなどの機密データを保存するシークレットとして環境変数を設定するためのものです。詳細については、「AWS Secrets Manager および Parameter Store での Elastic Beanstalk AWS Systems Manager の使用」を参照してください。

8. 環境変数名と環境変数値のペアを入力します。
9. さらに変数を追加する必要がある場合は、ステップ 6 からステップ 8 を繰り返します。
10. ページの最下部で [適用] を選択し変更を保存します。

ここで、サイトを環境にデプロイする準備が整いました。

アプリケーションをデプロイします

Rails で作成されたファイルを含む出典バンドルを作成します。次のコマンドでは、`rails-default.zip` という出典バンドルが作成されます。

```
~/eb-rails$ zip ../rails-default.zip -r * .[^.]*
```

ソースバンドルを Elastic Beanstalk にアップロードして、Rails を環境にデプロイします。

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

次のステップ

Rails の詳細については、rubyonrails.org を参照してください。

アプリケーションの開発が進むにつれ、.zip ファイルを手動で作成して Elastic Beanstalk コンソールにアップロードすることなく、環境を管理してアプリケーションをデプロイする方法が必要になります。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) には、コマンドラインインターフェイスからアプリケーションを作成、設定して、Elastic Beanstalk 環境にデプロイするための使いやすいコマンドが用意されています。

最後に、本稼働環境でアプリケーションを使用する予定の場合は、お客様の環境にカスタムドメイン名を設定し、安全な接続のために HTTPS を有効にすることが必要になります。

Elastic Beanstalk への Sinatra アプリケーションのデプロイ

このチュートリアルでは、シンプルな Sinatra ウェブアプリケーションを AWS Elastic Beanstalkにデプロイする方法について説明します。

前提条件

このチュートリアルでは、基本的な Elastic Beanstalk オペレーションと Elastic Beanstalk コンソールに関する知識があることを前提としています。まだ起動していない場合は、Elastic Beanstalk の使用を開始する方法について説明します。 の指示に従って、最初の Elastic Beanstalk 環境を起動します。

このガイドの手順に従うには、run command のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

Sinatra 2.1.0 には Ruby 2.3.0 以降が必要です。このチュートリアルでは、Ruby 3.0.2 と対応する Elastic Beanstalk プラットフォームバージョンを使用します。「Elastic Beanstalk 用の Ruby 開発環境の設定」の手順に従って Ruby をインストールします

Elastic Beanstalk 環境の起動

Elastic Beanstalk コンソールを使用して、Elastic Beanstalk 環境を作成します。[Ruby] プラットフォームを選択し、デフォルト設定およびサンプルコードを受け入れます。

環境を起動するには (コンソール)

1. 事前に設定されたリンク (console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced) を使用して、Elastic Beanstalk コンソールを開きます。
2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチを選択します。
3. アプリケーションコード として、サンプルアプリケーション を選択します。
4. 確認と起動 を選択します。
5. 使用できるオプションを確認します。使用する有効なオプションを選択し、準備ができたら [アプリケーションの作成] を選択します。

環境の作成の所要時間は約 5 分です。以下のリソースが作成されます。

- EC2 インスタンス – 選択したプラットフォームでウェブ・アプリケーションを実行するよう設定された Amazon Elastic Compute Cloud (Amazon EC2) 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、特定のソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、Apache または NGINX のいずれかをウェブアプリケーションの前にリバースプロキシとして配置します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供し、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- ロードバランサー – アプリケーションを実行するインスタンスにリクエストを分散するよう設定された Elastic Load Balancing ロードバランサー。ロードバランサーにより、インスタンスを直接インターネットに公開する必要もなくなります。
- ロードバランサーセキュリティグループ – ポート 80 上のインバウンドトラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、インターネットからの HTTP トラフィックが、ロードバランサーに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Auto Scaling グループ – インスタンスが終了されたか利用不可になった場合にそのインスタンスを置き換えるよう設定された Auto Scaling グループ。

- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷をモニタリングする 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック — Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

> **ドメインセキュリティ**
>
> Elastic Beanstalk アプリケーションのセキュリティを強化するため、elasticbeanstalk.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。Elastic Beanstalk アプリケーションのデフォルトドメイン名に機密性のある Cookie を設定する必要がある場合は、セキュリティ強化のため `__Host-` プレフィックスの付いた Cookie の使用をお勧めします。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御します。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。

これらのリソースはすべて Elastic Beanstalk によって管理されます。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

> **Note**
>
> Elastic Beanstalk が作成する Amazon S3 バケットは、環境間で共有され、環境の終了時に削除されません。詳細については、「Amazon S3 で Elastic Beanstalk を使用する」を参照してください。

ベーシック sinatra ウェブサイトの書き込み

sinatra アプリケーションを作成してデプロイするには

1. 次の内容で、config.ru という名前の設定ファイルを作成します。

   Example config.ru

   ```
   require './helloworld'
   run Sinatra::Application
   ```

2. 次の内容で、[helloworld.rb] という名前の Ruby コードファイルを作成します。

   Example helloworld.rb

   ```
   require 'sinatra'
   get '/' do
     "Hello World!"
   end
   ```

3. 次の内容で、[Gemfile] を作成します。

   Example Gemfile

   ```
   source 'https://rubygems.org'
   gem 'sinatra'
   gem 'puma'
   ```

4. `Gemfile.lock` を生成するバンドルインストールを実行します

   Example

   ```
   ~/eb-sinatra$ bundle install
   Fetching gem metadata from https://rubygems.org/....
   Resolving dependencies...
   Using bundler 2.2.22
   Using rack 2.2.3
   ...
   ```

5. Elastic Beanstalk が Ruby プラットフォームにアプリケーションを正常にデプロイするには、`Gemfile.lock`を更新する必要があります。`Gemfile.lock` のいくつかの依存関係は、

プラットフォーム固有かもしれません。したがって、**platform ruby**に`Gemfile.lock`を追加する必要があり、そうすると、必要な依存関係がすべてデプロイと共にインストールされます。

Example

```
~/eb-sinatra$ bundle lock --add-platform ruby
Fetching gem metadata from https://rubygems.org/....
Resolving dependencies...
Writing lockfile to /Users/janedoe/EBDPT/RubyApps/eb-sinatra/Gemfile.lock
```

6. 次のコンテンツで、 というプロファイルを作成します。

   Example [Procfile]

```
web: bundle exec puma -C /opt/elasticbeanstalk/config/private/pumaconf.rb
```

アプリケーションをデプロイします

出典ファイルを含む出典バンドルを作成します。次のコマンドでは、`sinatra-default.zip` という出典バンドルが作成されます。

```
~/eb-sinatra$ zip ../sinatra-default.zip -r * .[^.]*
```

ソースバンドルを Elastic Beanstalk にアップロードし、Sinatra を環境にデプロイします。

出典バンドルをデプロイするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. 環境の概要ページで、[Upload and deploy (アップロードとデプロイ)] を選択します。
4. 画面上のダイアログボックスを使用して、ソースバンドルをアップロードします。
5. [デプロイ] を選択します。
6. デプロイが完了したら、新しいタブのウェブサイトを開く、サイトの URL を選択できます。

クリーンアップ

デモコードの操作が完了したら、環境を終了できます。Elastic Beanstalk は、Amazon EC2 インスタンス、データベースインスタンス、ロードバランサー、セキュリティグループ、アラームなど、関連するすべての AWS リソースを削除します。

リソースを削除しても Elastic Beanstalk アプリケーションは削除されないため、アプリケーションの新しい環境はいつでも作成できます。

コンソールから Elastic Beanstalk 環境を終了するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

次のステップ

Sinatra の詳細については、sinatrarb.com を参照してください。

アプリケーションの開発が進むにつれ、.zip ファイルを手動で作成して Elastic Beanstalk コンソールにアップロードすることなく、環境を管理してアプリケーションをデプロイする方法が必要になります。Elastic Beanstalk コマンドラインインターフェイス (EB CLI) には、コマンドラインインターフェイスからアプリケーションを作成、設定して、Elastic Beanstalk 環境にデプロイするための使いやすいコマンドが用意されています。

最後に、本稼働環境でアプリケーションを使用する予定の場合は、お客様の環境にカスタムドメイン名を設定し、安全な接続のために HTTPS を有効にすることが必要になります。

Amazon RDS DB インスタンスを Ruby Elastic Beanstalk 環境に追加する

このトピックでは、Elastic Beanstalk コンソールを使用して Amazon RDS を作成する手順について説明します。Amazon Relational Database Service (Amazon RDS) DB インスタンスを使用して、アプリケーションによって収集および変更されたデータを保存することができます。データベースを環境に結合して Elastic Beanstalk で管理することも、分離したものとして作成して別のサービスで外部的に管理することもできます。これらの手順では、データベースは環境に結合され、Elastic

Beanstalk によって管理されます。Amazon RDS と Elastic Beanstalk の統合の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

セクション

- 環境に DB インスタンスを追加
- アダプタのダウンロード
- データベースへの接続

環境に DB インスタンスを追加

お客様の環境に DB インスタンスを追加するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [データベース] 設定カテゴリで、[編集] を選択します。
5. DB エンジンを選択して、ユーザー名とパスワードを入力します。
6. ページの最下部で [適用] を選択し変更を保存します。

DB インスタンスの追加には約 10 分かかります。環境の更新が完了すると、DB インスタンスのホスト名とその他の接続情報は以下の環境プロパティを通じてアプリケーションに使用できるようになります。

プロパティ名	説明	プロパティ値
`RDS_HOSTNAME`	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリティ)] タブ: [Endpoint (エンドポイント)]。
`RDS_PORT`	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。

プロパティ名	説明	プロパティ値
RDS_DB_NAME	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
RDS_USERNAME	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。
RDS_PASSWORD	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

Elastic Beanstalk 環境と結合したデータベースインスタンスの設定の詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

アダプタのダウンロード

プロジェクト gem ファイルにデータベースのアダプタを追加します。

Example Gemfile – MySQL を使用する Rails

```
source 'https://rubygems.org'
gem 'puma'
gem 'rails', '~> 6.1.4', '>= 6.1.4.1'
gem 'mysql2'
```

Ruby に共通のアダプタの Gem

- MySQL – `mysql2`
- PostgreSQL – `pg`
- Oracle – `activerecord-oracle_enhanced-adapter`
- SQL Server – `activerecord-sqlserver-adapter`

データベースへの接続

Elastic Beanstalk は、環境プロパティでアタッチされた DB インスタンスの接続情報を提供します。`ENV['VARIABLE']` を使用してプロパティを読み取り、データベース接続を設定します。

Example config/database.yml – Ruby on rails データベース設定 (MySQL)

```
production:
  adapter: mysql2
  encoding: utf8
  database: <%= ENV['RDS_DB_NAME'] %>
  username: <%= ENV['RDS_USERNAME'] %>
  password: <%= ENV['RDS_PASSWORD'] %>
  host: <%= ENV['RDS_HOSTNAME'] %>
  port: <%= ENV['RDS_PORT'] %>
```

Docker コンテナを使用した Elastic Beanstalk へのデプロイ

この章では、Elastic Beanstalk を使用して Docker コンテナからウェブアプリケーションをデプロイする方法について説明します。Docker コンテナは自己完結型で、これにはすべての設定情報と、ウェブアプリケーションが実行する必要のあるソフトウェアが含まれています。Docker コンテナを使用すると、独自のランタイム環境を定義できます。他の Elastic Beanstalk プラットフォームで通常はサポートされていない、独自のプログラミング言語とアプリケーションの従属関係 (パッケージマネージャやツールなど) を選択することもできます。

Docker の QuickStart の手順に従って Docker「Hello World」アプリケーションを作成し、EB CLI を使用して Elastic Beanstalk 環境にデプロイします。

トピック

- Elastic Beanstalk Docker プラットフォームブランチ
- Elastic Beanstalk Docker プラットフォームブランチを使用する
- Elastic Beanstalk で ECS マネージド Docker プラットフォームブランチを使用する
- イメージリポジトリを使用した認証
- Elastic Beanstalk Docker 環境の設定
- レガシープラットフォーム

Elastic Beanstalk Docker プラットフォームブランチ

Elastic Beanstalk の Docker プラットフォームでは、以下のプラットフォームブランチがサポートされています。

Docker running Amazon Linux 2 と Docker running AL2023

Elastic Beanstalk は Docker コンテナとソースコードを EC2 インスタンスにデプロイして管理します。これらのプラットフォームブランチはマルチコンテナをサポートします。Docker Compose ツールを使用すると、アプリケーションの設定、テスト、デプロイを簡素化できます。このプラットフォームブランチの詳細については、「the section called “Docker プラットフォームブランチ”」を参照してください。

Amazon Linux 2 上で動作する ECS と AL2023 上で動作する ECS

このブランチは、廃止されたプラットフォームブランチである (Amazon Linux AMI) 上で動作するマルチコンテナ Docker から AL2023/AL2 への移行パスが必要なお客様に提供されます。最新のプラッ

トフォームブランチは、廃止されたプラットフォームブランチのすべての機能をサポートします。ソースコードを変更する必要はありません。詳細については、「Elastic Beanstalk アプリケーションを AL1 の ECS マネージドマルチコンテナ Docker から Amazon Linux 2023 の ECS に移行する」を参照してください。ECS ベースのプラットフォームブランチで動作する Elastic Beanstalk 環境がない場合は、Docker Running on 64bit AL2023 プラットフォームブランチを使用することをお勧めします。これにより、よりシンプルなアプローチが提供され、必要なリソースが少なくなります。

これらの各プラットフォームブランチに関連付けられているソフトウェアコンポーネントバージョンのリストについては、「AWS Elastic Beanstalk プラットフォーム」ドキュメントの「Docker」を参照してください。

Amazon Linux AMI (AL1) で実行されている廃止されたプラットフォームブランチ

2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。廃止された各プラットフォームブランチと Amazon Linux 2 または Amazon Linux 2023 (推奨) 上で動作する最新のプラットフォームブランチへの移行パスの詳細については、次の各セクションを展開してください。

Docker (Amazon Linux AMI)

このプラットフォームブランチは、Dockerfile または `Dockerrun.aws.json` v1 定義で説明されている Docker イメージをデプロイできます。このプラットフォームブランチは、インスタンスごとに 1 つのコンテナのみを実行します。その後継プラットフォームブランチである Docker running on 64bit AL2023と Docker running on 64bit Amazon Linux 2 は、インスタンスごとに複数の Docker コンテナをサポートします。

サポートされている新しいプラットフォームブランチ Docker running on 64bit AL2023 を使用して環境を作成することをお勧めします。その後、アプリケーションを新しく作成した環境に移行できます。これらの環境の作成の詳細については、「the section called “Docker プラットフォームブランチ”」を参照してください。移行に関する詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

マルチコンテナ Docker (Amazon Linux AMI)

このプラットフォームブランチは、Amazon ECS を使用して、Elastic Beanstalk 環境で複数の Docker コンテナを Amazon ECS クラスターにデプロイするように調整します。現在この廃止されたプラットフォームブランチを使用している場合は、最新の Amazon Linux 2023 上で動作する ECS プラットフォームブランチに移行することをお勧めします。最新のプラットフォームブランチでは、こ

の廃止されたプラットフォームブランチのすべての機能がサポートされています。ソースコードを変更する必要はありません。詳細については、「Elastic Beanstalk アプリケーションを AL1 の ECS マネージドマルチコンテナ Docker から Amazon Linux 2023 の ECS に移行する」を参照してください。

事前設定済み Docker コンテナ

前述の Docker プラットフォームに加えて、Amazon Linux AMI オペレーティングシステム (AL1) 上で動作する Preconfigured Docker GlassFish プラットフォームブランチもあります。

このプラットフォームブランチは、Docker running on 64bit AL2023 と Docker running on 64bit Amazon Linux 2 プラットフォームブランチに置き換えられました。詳細については、「Docker プラットフォームへの GlassFish アプリケーションのデプロイ」を参照してください。

Elastic Beanstalk Docker プラットフォームブランチを使用する

このセクションでは、AL2 または AL2023 を実行する Docker の Elastic Beanstalk プラットフォームブランチのいずれかを使用して、Docker イメージを起動用に準備する方法について説明します。

Docker の QuickStart の手順に従って Docker「Hello World」アプリケーションを作成し、EB CLI を使用して Elastic Beanstalk 環境にデプロイします。

トピック

- QuickStart: Elastic Beanstalk に Docker アプリケーションをデプロイする
- QuickStart: Docker Compose アプリケーションを Elastic Beanstalk にデプロイする
- Elastic Beanstalk へのデプロイ用に Docker イメージを準備する

QuickStart: Elastic Beanstalk に Docker アプリケーションをデプロイする

この QuickStart チュートリアルでは、Docker アプリケーションを作成し、 AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。

> **Note**
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: Docker アプリケーションとコンテナを作成する
- ステップ 2: アプリケーションをローカルに実行する
- ステップ 3: EB CLI を使用して Docker アプリケーションをデプロイする
- ステップ 4: Elastic Beanstalk でアプリケーションを実行する
- ステップ 5：クリーンアップ
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

AWS アカウントを既にお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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 Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

EB CLI

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

Docker

このチュートリアルを実行するには、作業用 Docker がローカルにインストールされている必要があります。詳細については、Docker ドキュメントウェブサイトの「Get Docker」を参照してください。

次のコマンドを実行して、Docker デーモンが起動され実行されていることを確認します。

```
~$ docker info
```

ステップ 1: Docker アプリケーションとコンテナを作成する

この例では、Elastic Beanstalk への Flask アプリケーションのデプロイ でも参照されているサンプル Flask アプリケーションの Docker イメージを作成します。

アプリケーションは、2 つのファイルで構成されます。

- `app.py` – コンテナで実行されるコードを含む Python ファイル。
- `Dockerfile` – イメージをビルドする Dockerfile。

両方のファイルをディレクトリのルートに置きます。

```
~/eb-docker-flask/
|-- Dockerfile
|-- app.py
```

`Dockerfile` に次の内容を追加します。

Example **`~/eb-docker-flask/Dockerfile`**

```
FROM public.ecr.aws/docker/library/python:3.12
COPY . /app
WORKDIR /app
RUN pip install Flask==3.1.1
EXPOSE 5000
CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

`app.py` ファイルに次の内容を追加します。

Example **~/eb-docker-flask/app.py**

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello Elastic Beanstalk! This is a Docker application'
```

docker build コマンドを使用してコンテナイメージをローカルにビルドし、イメージに eb-docker-flask をタグ付けします。コマンドの最後にあるピリオド (.) は、パスがローカルディレクトリであることを指定しています。

```
~/eb-docker-flask$ docker build -t eb-docker-flask .
```

ステップ 2: アプリケーションをローカルに実行する

docker run コマンドを使用してコンテナを実行します。コマンドは実行中のコンテナの ID を出力します。-d オプションはバックグラウンドモードで docker を実行します。-p オプションは、ポート 5000 でアプリケーションを公開します。Elastic Beanstalk は、デフォルトで Docker プラットフォームのポート 5000 へのトラフィックを提供します。

```
~/eb-docker-flask$ docker run -dp 127.0.0.1:5000:5000 eb-docker-flask
```

ブラウザで http://127.0.0.1:5000/ にアクセスします。次のテキストが表示されます。「Hello Elastic Beanstalk! これは Docker アプリケーションです」。

docker kill コマンドを実行してコンテナを終了します。

```
~/eb-docker-flask$ docker kill container-id
```

ステップ 3: EB CLI を使用して Docker アプリケーションをデプロイする

次のコマンドを実行して、このアプリケーションの Elastic Beanstalk 環境を作成します。

環境を作成し、Docker アプリケーションをデプロイするには

1. eb init コマンドを使用して EB CLI リポジトリを初期化します。

   ```
   ~/eb-docker-flask$ eb init -p docker docker-tutorial --region us-east-2
   ```

```
Application docker-tutorial has been created.
```

このコマンドは、`docker-tutorial` という名前のアプリケーションを作成し、ローカルリポジトリを設定して最新の Docker プラットフォームバージョンで環境を作成します。

2. (オプション) eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH を使用して connect できるようにします。

```
~/eb-docker-flask$ eb init
Do you want to set up SSH for your instances?
(y/n): y
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従ってキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。

3. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。Elastic Beanstalk は、アプリケーションの zip ファイルを自動的にビルドし、ポート 5000 で起動します。

```
~/eb-docker-flask$ eb create docker-tutorial
```

Elastic Beanstalk が環境を作成するのに約 5 分かかります。

ステップ 4: Elastic Beanstalk でアプリケーションを実行する

環境を作成するプロセスが完了したら、eb open でウェブサイトを開きます。

```
~/eb-docker-flask$ eb open
```

お疲れ様でした。Elastic Beanstalk で Docker アプリケーションをデプロイしました。これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ Window が開きます。

ステップ 5：クリーンアップ

アプリケーションでの作業が終了したら、環境を終了できます。Elastic Beanstalk は、環境に関連付けられているすべての AWS リソースを終了します。

EB CLI を使用して Elastic Beanstalk 環境を終了するには、次のコマンドを実行します。

```
~/eb-docker-flask$ eb terminate
```

AWS アプリケーションの リソース

1 つのインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありません。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

1 つか 2 つのサンプルアプリケーションをデプロイし、ローカルで Docker アプリケーションを開発して実行する準備が整ったら、「Elastic Beanstalk へのデプロイ用に Docker イメージを準備する」を参照します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用してサンプルアプリケーションを起動することもできます。詳細な手順については、このガイドの「開始方法」の章の「サンプルアプリケーションを作成する」を参照してください。

QuickStart: Docker Compose アプリケーションを Elastic Beanstalk にデプロイする

この QuickStart チュートリアルでは、複数コンテナの Docker Compose アプリケーションを作成し、 AWS Elastic Beanstalk 環境にデプロイするプロセスについて説明します。nginx リバースプロキシを使用して Flask ウェブアプリケーションを作成し、Docker Compose が複数のコンテナのオーケストレーションを簡素化する方法を示します。

> **Note**
>
> チュートリアルの例は、デモンストレーションを目的としています。本番トラフィックにアプリケーションを使用しないでください。

セクション

- AWS アカウント
- 前提条件
- ステップ 1: Docker Compose アプリケーションを作成する

- ステップ 2: アプリケーションをローカルに実行する
- ステップ 3: EB CLI を使用して Docker Compose アプリケーションをデプロイする
- ステップ 4: Elastic Beanstalk でアプリケーションをテストする
- ステップ 5：クリーンアップ
- AWS アプリケーションの リソース
- 次のステップ
- Elastic Beanstalk コンソールでデプロイする

AWS アカウント

まだ AWS のお客様でない場合は、 AWS アカウントを作成する必要があります。サインアップすると、Elastic Beanstalk やその他の必要な AWS サービスにアクセスできます。

AWS アカウントを既にお持ちの場合は、「」に進むことができます前提条件。

AWS アカウントを作成する

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 Identity Center、 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 Management Console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 Identity Center ユーザーガイド」の「AWS IAM Identity Center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 Identity Center 「 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 Identity Center ユーザーガイド」の「権限設定を作成する」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 Identity Center ユーザーガイド」の「グループの結合」を参照してください。

前提条件

このガイドの手順に従うには、run commands のためのコマンドラインターミナルまたはシェルが必要になります。コマンドは、該当する場合、プロンプト記号 ($) と現在のディレクトリの名前が前に付けられて、リストに示されます。

```
~/eb-project$ this is a command
this is output
```

Linux および macOS では、任意のシェルとパッケージ管理者を使用できます。Windows では、Linux 用の Windows サブシステムをインストールして、Ubuntu および Bash の Windows に統合されたバージョンを入手できます。

EB CLI

このチュートリアルでは、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用します。EB CLI をインストールおよび設定する手順の詳細については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」および「EB CLI の設定」を参照してください。

Docker と Docker Compose

このチュートリアルに従うには、Docker と Docker Compose のローカルインストールが必要です。詳細については、Docker ドキュメントウェブサイトの「Get Docker and Install Docker Compose」を参照してください。

次のコマンドを実行して、Docker と Docker Compose がインストールされ、実行されていることを確認します。

```
~$ docker info
~$ docker compose version
```

ステップ 1: Docker Compose アプリケーションを作成する

この例では、Flask ウェブアプリケーションと nginx リバースプロキシで構成される Docker Compose を使用してマルチコンテナアプリケーションを作成します。これは、Docker Compose が連携する複数のコンテナのオーケストレーションを簡素化する方法を示しています。

アプリケーションには、Elastic Beanstalk が nginx プロキシから詳細なアプリケーションメトリクスを収集できるようにするヘルスモニタリング設定が含まれています。

アプリケーションは次の構造で構成されます。

```
~/eb-docker-compose-flask/
|-- docker-compose.yml
|-- web/
|   |-- Dockerfile
|   |-- app.py
|   `-- requirements.txt
|-- proxy/
|   |-- Dockerfile
|   `-- nginx.conf
`-- .platform/
    `-- hooks/
        `-- postdeploy/
            `-- 01_setup_healthd_permissions.sh
```

ディレクトリ構造を作成し、次のファイルを追加します。

まず、サービスとその関係を定義するメイン`docker-compose.yml`ファイルを作成します。

Example **`~/eb-docker-compose-flask/docker-compose.yml`**

```
services:
  web:
    build: ./web
    expose:
      - "5000"

  nginx-proxy:
    build: ./proxy
```

```
    ports:
      - "80:80"
    volumes:
      - "/var/log/nginx:/var/log/nginx"
    depends_on:
      - web
```

web ディレクトリに Flask ウェブアプリケーションを作成します。app.py ファイルに次の内容を追加します。

Example **~/eb-docker-compose-flask/web/app.py**

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello Elastic Beanstalk! This is a Docker Compose application'
```

ウェブサービス に次のコンテンツを追加しますDockerfile。

Example **~/eb-docker-compose-flask/web/Dockerfile**

```
FROM public.ecr.aws/docker/library/python:3.12
COPY . /app
WORKDIR /app
RUN pip install Flask==3.1.1
EXPOSE 5000
CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

proxy ディレクトリに nginx リバースプロキシを作成します。nginx.conf ファイルに次の内容を追加します。

この設定には、Elastic Beanstalk が詳細なアプリケーションメトリクスを収集できるようにするヘルスモニタリング設定が含まれています。ヘルスモニタリングログ形式のカスタマイズの詳細については、「」を参照してください拡張ヘルスログ形式。

Example **~/eb-docker-compose-flask/proxy/nginx.conf**

```
events {
    worker_connections 1024;
}
```

```
http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    map $http_upgrade $connection_upgrade {
        default       "upgrade";
    }

    # Health monitoring log format for Elastic Beanstalk
    log_format healthd
 '$msec"$uri"$status"$request_time"$upstream_response_time"$http_x_forwarded_for';

    upstream flask_app {
        server web:5000;
    }

    server {
        listen 80 default_server;
        root /usr/share/nginx/html;

        # Standard access log
        access_log /var/log/nginx/access.log;

        # Health monitoring log for Elastic Beanstalk
        if ($time_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2})") {
            set $year $1;
            set $month $2;
            set $day $3;
            set $hour $4;
        }
        access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour
 healthd;

        location / {
            proxy_pass http://flask_app;
            proxy_http_version    1.1;

            proxy_set_header    Connection             $connection_upgrade;
            proxy_set_header    Upgrade                $http_upgrade;
            proxy_set_header    Host                   $host;
            proxy_set_header    X-Real-IP              $remote_addr;
            proxy_set_header    X-Forwarded-For        $proxy_add_x_forwarded_for;
        }
    }
```

```
}
```

プロキシサービス に次のコンテンツを追加しますDockerfile。

Example **~/eb-docker-compose-flask/proxy/Dockerfile**

```
FROM public.ecr.aws/nginx/nginx:alpine
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
```

最後に、プラットフォームフックスクリプトを作成して、ヘルスモニタリングに必要なログディレクトリとアクセス許可を設定します。プラットフォームフックを使用すると、デプロイプロセス中にカスタムスクリプトを実行できます。プラットフォームフックの詳細については、「」を参照してください[プラットフォームフック](#)。

Example **~/eb-docker-compose-flask/.platform/hooks/postdeploy/01_setup_healthd_permissions.sh**

```
#!/bin/bash
set -ex

NGINX_CONTAINER=$(docker ps --filter "name=nginx-proxy" -q)

if [ -z "$NGINX_CONTAINER" ]; then
    echo "Error: No nginx-proxy container found running"
    exit 1
fi

NGINX_UID=$(docker exec ${NGINX_CONTAINER} id -u nginx)
NGINX_GID=$(docker exec ${NGINX_CONTAINER} id -g nginx)

mkdir -p /var/log/nginx/healthd
chown -R ${NGINX_UID}:${NGINX_GID} /var/log/nginx
```

ステップ 2: アプリケーションをローカルに実行する

[docker compose up](#) コマンドを使用して、マルチコンテナアプリケーションをローカルで構築して実行します。Docker Compose は両方のコンテナイメージを構築し、 `docker-compose.yml` ファイルで定義されたサービスを開始します。

```
~/eb-docker-compose-flask$ docker compose up --build
```

--build オプションにより、Docker Compose はサービスを開始する前にコンテナイメージを構築できます。ウェブサービスと nginx-proxy サービスの両方が起動していることを示す出力が表示されます。

ブラウザ`http://localhost`で に移動します。次のテキストが表示されます。「Hello Elastic Beanstalk! これは Docker Compose アプリケーションです」。nginx プロキシは、ポート 80 でリクエストを受け取り、ポート 5000 で実行されている Flask アプリケーションに転送します。

テストが完了したら、Ctrl+Cターミナルで を押してアプリケーションを停止するか、別のターミナルで次のコマンドを実行します。

```
~/eb-docker-compose-flask$ docker compose down
```

ステップ 3: EB CLI を使用して Docker Compose アプリケーションをデプロイする

次のコマンドを実行して、このアプリケーションの Elastic Beanstalk 環境を作成します。

環境を作成し、Docker Compose アプリケーションをデプロイするには

1. eb init コマンドを使用して EB CLI リポジトリを初期化します。

   ```
   ~/eb-docker-compose-flask$ eb init -p docker docker-compose-tutorial --region us-east-2
   Application docker-compose-tutorial has been created.
   ```

   このコマンドは、`docker-compose-tutorial` という名前のアプリケーションを作成し、ローカルリポジトリを設定して最新の Docker プラットフォームバージョンで環境を作成します。

2. (オプション) eb init を再度実行してデフォルトのキーペアを設定し、アプリケーションを実行している EC2 インスタンスに SSH を使用して connect できるようにします。

   ```
   ~/eb-docker-compose-flask$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

1 つのキーペアがすでにある場合はそれを選択するか、またはプロンプトに従ってキーペアを作成します。プロンプトが表示されないか設定を後で変更する必要がない場合は、eb init -i を実行します。

3. 環境を作成し、eb create を使用してそこにアプリケーションをデプロイします。Elastic Beanstalk は`docker-compose.yml`ファイルを自動的に検出し、マルチコンテナアプリケーションをデプロイします。

```
~/eb-docker-compose-flask$ eb create docker-compose-tutorial
```

Elastic Beanstalk が環境を作成し、マルチコンテナアプリケーションをデプロイするまでに約 5 分かかります。

ステップ 4: Elastic Beanstalk でアプリケーションをテストする

環境を作成するプロセスが完了したら、eb open でウェブサイトを開きます。

```
~/eb-docker-compose-flask$ eb open
```

素晴らしい！Elastic Beanstalk でマルチコンテナの Docker Compose アプリケーションをデプロイしました。これにより、アプリケーション用に作成されたドメイン名を使用してブラウザ Window が開きます。nginx リバースプロキシを介して提供される Flask アプリケーションからのメッセージが表示されます。

ステップ 5：クリーンアップ

アプリケーションでの作業が終了したら、環境を終了できます。Elastic Beanstalk は、環境に関連付けられているすべての AWS リソースを終了します。

EB CLI を使用して Elastic Beanstalk 環境を終了するには、次のコマンドを実行します。

```
~/eb-docker-compose-flask$ eb terminate
```

AWS アプリケーションの リソース

複数のコンテナを実行する単一のインスタンスアプリケーションを作成しました。1 つの EC2 インスタンスを持つ簡単なサンプルアプリケーションとして動作するため、ロードバランシングや自動スケーリングは必要ありません。単一インスタンスアプリケーションの場合、Elastic Beanstalk は次の AWS リソースを作成します。

- EC2 インスタンス – 選択したプラットフォームでウェブアプリケーションを実行するよう設定された Amazon EC2 仮想マシン。

  各プラットフォームは、それぞれ特定の言語バージョン、フレームワーク、ウェブコンテナ、またはそれらの組み合わせをサポートするための、さまざまなソフトウェア、設定ファイル、スクリプトを実行します。ほとんどのプラットフォームでは、ウェブアプリケーションの前にウェブトラフィックを処理するリバースプロキシとして Apache または nginx のいずれかを使用します。そのプロキシがリクエストをアプリケーションに転送し、静的アセットを提供して、アクセスログとエラーログを生成します。
- インスタンスセキュリティグループ – ポート 80 上の受信トラフィックを許可するよう設定された Amazon EC2 セキュリティグループ。このリソースでは、ロードバランサーからの HTTP トラフィックが、ウェブ・アプリケーションを実行している EC2 インスタンスに達することができます。デフォルトでは、トラフィックは他のポート上で許可されません。
- Amazon S3 バケット – Elastic Beanstalk の使用時に作成されるソースコード、ログ、その他のアーティファクトの保存場所。
- Amazon CloudWatch アラーム – 環境内のインスタンスの負荷を監視する 2 つの CloudWatch アラーム。負荷が高すぎる、または低すぎる場合にトリガーされます。アラームがトリガーされると、Auto Scaling グループはレスポンスとしてスケールアップまたはダウンを行います。
- AWS CloudFormation スタック – Elastic Beanstalk は AWS CloudFormation を使用して環境内のリソースを起動し、設定変更を伝達します。リソースは、AWS CloudFormation コンソールに表示できるテンプレートで定義されます。
- ドメイン名 – ウェブ・アプリケーションまでのルートとなるドメイン名であり、*subdomain*.*region*.elasticbeanstalk.com の形式です。

Elastic Beanstalk は、これらのリソースをすべて管理します。環境を終了すると、Elastic Beanstalk は含まれているすべてのリソースを終了します。Docker Compose アプリケーションは、単一の EC2 インスタンスで複数のコンテナを実行し、Elastic Beanstalk がオーケストレーションを自動的に処理します。

次のステップ

アプリケーションを実行する環境を手に入れた後、アプリケーションの新しいバージョンや、異なるアプリケーションをいつでもデプロイできるようになります。新しいアプリケーションバージョンのデプロイは、プロビジョニングや EC2 インスタンスの再開が必要ないため、非常に素早く行えます。Elastic Beanstalk コンソールを使用して新しい環境を調べることもできます。詳細な手順については、このガイドの「開始方法」の章の「環境を探索する」を参照してください。

サンプルアプリケーションを 1 つか 2 つデプロイし、Docker Compose アプリケーションの開発と実行をローカルで開始する準備ができたら、「」を参照してください[Elastic Beanstalk へのデプロイ用に Docker イメージを準備する](#)。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールを使用して Docker Compose アプリケーションを起動することもできます。ファイル`docker-compose.yml`と関連するすべてのディレクトリとファイルを含む ZIP ファイルを作成し、新しいアプリケーションを作成するときにアップロードします。詳細な手順については、このガイドの「開始方法」の章の「[サンプルアプリケーションを作成する](#)」を参照してください。

Elastic Beanstalk へのデプロイ用に Docker イメージを準備する

このセクションでは、AL2 または AL2023 を実行している Docker のプラットフォームブランチのいずれかを使用して Elastic Beanstalk にデプロイするための Docker イメージを準備する方法について説明します。必要な設定ファイルは、イメージがローカルかリモートか、Docker Compose を使用しているかどうかによって異なります。

> **Note**
>
> Docker 環境を起動する手順の例については、「[Docker の QuickStart](#)」トピックを参照してください。

トピック

- [Elastic Beanstalk で Docker Compose を使用してイメージを管理する](#)
- [Elastic Beanstalk で Docker Compose を使用せずにイメージを管理する](#)
- [Dockerfile を使用したカスタムイメージの構築](#)

Elastic Beanstalk で Docker Compose を使用してイメージを管理する

Docker Compose を使用して、1 つの YAML ファイルでさまざまなサービスを管理できます。Docker Compose の詳細については、Docker ウェブサイトの「[Why use Compose?](#)」を参照してください。

- `docker-compose.yml` を作成します。このファイルは、Docker Compose を使用して Elastic Beanstalk でアプリケーションを管理している場合に必要です。すべてのデプロイのソースがパ

ブリックリポジトリ内のイメージである場合、他の設定ファイルは不要です。デプロイのソースイメージがプライベートリポジトリにある場合は、追加の設定を行う必要があります。詳細については、「プライベートリポジトリからのイメージの使用」を参照してください。`docker-compose.yml` ファイルの詳細については、Docker ウェブサイトの「Compose file reference」を参照してください。

- `Dockerfile` はオプションです。Elastic Beanstalk でローカルのカスタムイメージをビルドして実行する場合は、このファイルを作成します。`Dockerfile` の詳細については、Docker ウェブサイトの「Dockerfile reference」を参照してください。
- `.zip` ファイルの作成が必要になる場合があります。`Dockerfile` ファイルのみを使用してアプリケーションをデプロイする場合、このファイルを作成する必要はありません。追加の設定ファイルを使用する場合は、.zip ファイルには、`Dockerfile`、`docker-compose.yml` ファイル、アプリケーションファイル、およびアプリケーションファイルの依存関係を含める必要があります。`Dockerfile` と `docker-compose.yml` は、.zip アーカイブのルート、つまり最上位レベルにある必要があります。EB CLI を使用してアプリケーションをデプロイする場合には、.zip ファイルが自動的に作成されます。

Docker Compose の詳細およびインストール方法については、Docker サイトの「Docker Compose の概要」および「Docker Compose のインストール」を参照してください。

Elastic Beanstalk で Docker Compose を使用せずにイメージを管理する

Docker Compose を使用して Docker イメージを管理していない場合は、`Dockerfile`、`Dockerrun.aws.json` ファイル、またはその両方を設定する必要があります。

- `Dockerfile` を作成し、Elastic Beanstalk でカスタムイメージをローカルにビルドして実行します。
- `Dockerrun.aws.json v1` ファイルを作成して、ホストされたレポジトリから Elastic Beanstalk に Docker イメージをデプロイします。
- `.zip` ファイルの作成が必要になる場合があります。`Dockerfile` または `Dockerrun.aws.json` のいずれか 1 つのファイルのみを使用する場合は、.zip ファイルを作成する必要はありません。両方のファイルを使用する場合は、.zip ファイルが必要です。.zip ファイルには、アプリケーションファイルとアプリケーションファイルの依存関係を含むファイルに加えて、`Dockerfile` と `Dockerrun.aws.json` の両方が含まれている必要があります。EB CLI を使用してアプリケーションをデプロイする場合には、`.zip` ファイルが自動的に作成されます。

Dockerrun.aws.json v1 設定ファイル

`Dockerrun.aws.json` ファイルは、リモート Docker イメージを Elastic Beanstalk アプリケーションとしてデプロイする方法を記述します。この JSON ファイルは Elastic Beanstalk に固有です。ホストされたレポジトリで使用できるイメージでアプリケーションが実行される場合、`Dockerrun.aws.json v1` ファイルでイメージを指定し、`Dockerfile` を省略できます。

> **Dockerrun.aws.json** バージョン
>
> `AWSEBDockerrunVersion` パラメータは、`Dockerrun.aws.json` ファイルのバージョンを示します。
>
> - Docker AL2 および AL2023 プラットフォームでは、次のバージョンのファイルを使用します。
> - `Dockerrun.aws.json v3` – Docker Compose を使用する環境。
> - `Dockerrun.aws.json v1` – Docker Compose を使用しない環境。
> - Amazon Linux 2 上で実行される ECS と AL2023 上で実行される ECS は `Dockerrun.aws.json v2` ファイルを使用します。廃止されたプラットフォームである ECS - マルチコンテナ Docker Amazon Linux AMI (AL1) も同じバージョンを使用していました。

Dockerrun.aws.json v1

`Dockerrun.aws.json v1` ファイルの有効なキーと値には、以下のオペレーションが含まれます。

AWSEBDockerrunVersion

(必須) Docker Compose を使用してイメージを管理していない場合は、バージョン番号 1 を指定します。

認証

(プライベートリポジトリの場合にのみ必須) `.dockercfg` ファイルを保存する Amazon S3 オブジェクトを指定します。

この章で後述する「プライベートリポジトリからのイメージの使用」の「イメージリポジトリを使用した認証」を参照してください。

イメージ

Docker コンテナを作成するときにベースとなる既存の Docker リポジトリの Docker ベースイメージを指定します。Name キーの値を、Docker Hub 上のイメージの場合は *<organization>/<image name>* 形式で、その他のサイトの場合は *<site>/<organization name>/<image name>* 形式で指定します。

`Dockerrun.aws.json` ファイルでイメージを指定すると、Elastic Beanstalk 環境内の各インスタンスが `docker pull` を実行してイメージを実行します。必要に応じて Update キーを含めます。デフォルト値は `true` であり、これはリポジトリをチェックし、イメージに対する更新を検出して、キャッシュされているイメージを上書きするように Elastic Beanstalk に指示します。

`Dockerfile` を使用するときは、`Dockerrun.aws.json` ファイルで Image キーを指定しないでください。Elastic Beanstalk は、存在する場合は `Dockerfile` に示されているイメージを常に構築して使用します。

ポート

(Image キーを指定する場合は必須) Docker コンテナで公開するポートをリストアップします。Elastic Beanstalk は、ContainerPort の値を使用して、ホストで実行されているリバースプロキシに Docker コンテナを接続します。

複数のコンテナポートを指定できますが、Elastic Beanstalk では最初のポートのみが使用されます。このポートを使用して、コンテナをホストのリバースプロキシに接続し、公衆インターネットからのリクエストをルーティングします。`Dockerfile` を使用している場合、最初の ContainerPort の値は、`Dockerfile` の EXPOSE リストの最初のエントリに一致する必要があります。

必要に応じて、HostPort でポートのリストを指定することができます。HostPort エントリは、ContainerPort の値がマッピングされるホストポートを指定します。HostPort 値を指定しなかった場合、デフォルトで ContainerPort 値に設定されます。

```
{
  "Image": {
    "Name": "image-name"
  },
  "Ports": [
    {
      "ContainerPort": 8080,
      "HostPort": 8000
    }
```

```
  ]
}
```

ボリューム

EC2 インスタンスのボリュームを Docker コンテナにマッピングします。1 つ以上のボリューム配列をマッピング対象として指定します。

```
{
  "Volumes": [
    {
      "HostDirectory": "/path/inside/host",
      "ContainerDirectory": "/path/inside/container"
    }
  ]
...
```

ログ収集

アプリケーションがログを書き込むコンテナ内のディレクトリを指定します。ログ末尾やバンドルログをリクエストすると、Elastic Beanstalk によって Amazon S3 にこのディレクトリ内のログがすべてアップロードされます。このディレクトリ内の `rotated` という名前のフォルダに対してログをローテーションさせる場合は、ローテーションさせたログを保管用に Amazon S3 にアップロードするように、Elastic Beanstalk を設定することもできます。詳細については、「」を参照してください[Elastic Beanstalk 環境の Amazon EC2 インスタンスからのログの表示](#)

コマンド

コンテナで実行するコマンドを指定します。エントリポイントを指定した場合、コマンドは引数としてエントリポイントに追加されます。詳細については、Docker ドキュメントの[CMD](#) を参照してください。

エントリポイント

コンテナの開始時に実行するデフォルトのコマンドを指定します。詳細については、Docker ドキュメントの [docker ps](#) を参照してください。

以下のスニペットは、1 つのコンテナの `Dockerrun.aws.json` ファイルの構文を示す例です。

```
{
```

```
  "AWSEBDockerrunVersion": "1",
  "Image": {
    "Name": "janedoe/image",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": "1234"
    }
  ],
  "Volumes": [
    {
      "HostDirectory": "/var/app/mydb",
      "ContainerDirectory": "/etc/mysql"
    }
  ],
  "Logging": "/var/log/nginx",
  "Entrypoint": "/app/bin/myapp",
  "Command": "--argument"
}>
```

`Dockerrun.aws.json` ファイルのみ、または `.zip` と `Dockerrun.aws.json` ファイルの両方を含んだ `Dockerfile` アーカイブを Elastic Beanstalk に提供できます。両方のファイルを提供する場合、`Dockerfile` は Docker イメージを記述し、`Dockerrun.aws.json` ファイルはデプロイに関する追加情報を提供します。これについては後ほど説明します。

> **Note**
>
> 2 つのファイルは、`.zip` アーカイブのルートまたは最上位レベルにある必要があります。ファイルを含むディレクトリからアーカイブを構築しないでください。代わりに、そのディレクトリに移動し、そこでアーカイブを構築します。
> 両方のファイルを提供する場合は、`Dockerrun.aws.json` ファイルにイメージを指定しないでください。Elastic Beanstalk は `Dockerfile` で記述されているイメージを構築および使用し、`Dockerrun.aws.json` ファイルに指定されているイメージを無視します。

Dockerfile を使用したカスタムイメージの構築

リポジトリでホストされている既存のイメージがない場合は、`Dockerfile` を作成する必要があります。

以下のスニペットは Dockerfile の例です。「Docker の QuickStart」の手順に従う場合は、この手順の Dockerfile をそのままアップロードできます。この Dockerfile を使用する場合、Elastic Beanstalk はゲーム 2048 を実行します。

Dockerfile に含めることができる命令の詳細については、Docker ウェブサイトの Dockerfile Reference を参照してください。

```
FROM ubuntu:12.04

RUN apt-get update
RUN apt-get install -y nginx zip curl

RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN curl -o /usr/share/nginx/www/master.zip -L https://codeload.github.com/
gabrielecirulli/2048/zip/master
RUN cd /usr/share/nginx/www/ && unzip master.zip && mv 2048-master/* . && rm -rf 2048-
master master.zip

EXPOSE 80

CMD ["/usr/sbin/nginx", "-c", "/etc/nginx/nginx.conf"]
```

> **Note**
>
> 単一の Dockerfile からマルチステージビルドを実行して、より小さなサイズのイメージを生成し、複雑さを大幅に軽減できます。詳細については、Docker ドキュメントのウェブサイトのマルチステージビルドを使用するを参照してください。

Elastic Beanstalk で ECS マネージド Docker プラットフォームブランチを使用する

このトピックでは、Amazon Linux 2 および Amazon Linux 2023 用の Elastic Beanstalk ECS マネージド Docker プラットフォームブランチの概要について説明します。また、Docker ECS マネージドプラットフォームに固有の設定情報も提供します。

AL1 上のマルチコンテナ Docker からの移行

2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。この章では、この廃止されたプラットフォームの設定

情報を提供しますが、サポートされている最新のプラットフォームブランチに移行することを強くお勧めします。現在、廃止された AL1 上で動作するマルチコンテナ Docker プラットフォームブランチを使用している場合は、最新の AL2023 上で動作する ECS プラットフォームブランチに移行できます。最新のプラットフォームブランチでは、廃止されたプラットフォームブランチのすべての機能がサポートされています。ソースコードを変更する必要はありません。詳細については、「Elastic Beanstalk アプリケーションを AL1 の ECS マネージドマルチコンテナ Docker から Amazon Linux 2023 の ECS に移行する」を参照してください。

ECS マネージド Docker プラットフォームの概要

Elastic Beanstalk は、Amazon Elastic Container Service (Amazon ECS) を使用して、ECS マネージド Docker 環境へのコンテナのデプロイを調整します。Amazon ECS は、Docker コンテナを実行するインスタンスのクラスターを管理するためのツールを供給します。Elastic Beanstalk は、クラスター作成、タスクの定義と実行のような Amazon ECS のタスクを処理します。環境内のインスタンスはそれぞれ、`Dockerrun.aws.json` v2 ファイルで定義される同じセットのコンテナを実行します。Docker を最大限に活用するため、Elastic Beanstalk では、Amazon EC2 インスタンスが複数の Docker コンテナを並行して実行できる環境を作成することができます。

次の図は、Auto Scaling グループの各 Amazon EC2 インスタンスで実行される 3 つの Docker コンテナで設定された Elastic Beanstalk 環境の例を示しています。

Elastic Beanstalk によって作成された Amazon ECS リソース

ECS マネージド Docker プラットフォームを使用して環境を作成する場合、環境の構築中に Elastic Beanstalk によって自動的に複数の Amazon Elastic Container Service リソースが作成および設定されます。これにより、各 Amazon EC2 インスタンスに必要なコンテナが作成されます。

- Amazon ECS クラスター – Amazon ECS のコンテナインスタンスはクラスターに整理されます。Elastic Beanstalk とともに使用すると、ECS マネージド Docker 環境ごとに必ず 1 つのクラスターが作成されます。ECS クラスターには、Auto Scaling グループのキャパシティープロバイダーやその他のリソースも含まれています。
- Amazon ECS タスク定義 – Elastic Beanstalk は、プロジェクト内の `Dockerrun.aws.json` v2 ファイルを使用して、環境内のコンテナインスタンスの設定に使用される Amazon ECS タスク定義を生成します。
- Amazon ECS タスク – Elastic Beanstalk は Amazon ECS と通信して、環境の各インスタンスでタスクを実行し、コンテナのデプロイを調整します。スケーラブルな環境では、Elastic Beanstalk はインスタンスがクラスターに追加されるたびに新しいタスクを開始します。

- Amazon ECS コンテナエージェント – エージェントは環境のインスタンスの Docker コンテナで実行されます。エージェントは Amazon ECS サービスをポーリングし、タスクの実行を待ちます。
- Amazon ECS データボリューム – `Dockerrun.aws.jsonv2` で定義したボリュームに加えて、Elastic Beanstalk はボリューム定義をタスク定義に挿入してログ収集を容易にします。

  Elastic Beanstalk はコンテナインスタンスにログボリュームを作成します。コンテナごとに 1 つ、場所は `/var/log/containers/containername` です。これらのボリュームの名前は `awseb-logs-containername` で、マウントするコンテナごとに指定されます。このマウント方法の詳細については、「コンテナの定義形式」を参照してください。

Amazon ECS リソースの詳細については、「Amazon Elastic Container Service デベロッパーガイド」を参照してください。

`Dockerrun.aws.json` v2 ファイル

コンテナインスタンスには、`Dockerrun.aws.json` という名前の設定ファイルが必要です。コンテナインスタンスとは、Elastic Beanstalk 環境で ECS マネージド Docker を実行する Amazon EC2 インスタンスです。このファイルは Elastic Beanstalk に固有であり、単独で、またはソースバンドルでソースコードやコンテンツと組み合わせて使用して、Docker プラットフォーム上に環境を作成することができます。

> **Note**
>
> `Dockerrun.aws.json` のバージョン 2 では、Amazon EC2 インスタンスごとに複数のコンテナのサポートが追加され、ECS マネージド Docker プラットフォームとの組み合わせでのみ使用できます。形式は、ECS によって管理されていない Docker プラットフォームブランチをサポートする他の設定ファイルバージョンとは大きく異なります。

更新された形式とサンプルファイルの詳細については、「`Dockerrun.aws.json v2`」を参照してください。

Docker イメージ

Elastic Beanstalk の ECS マネージド Docker プラットフォームでは、Elastic Beanstalk 環境を作成する前に、イメージを事前に作成し、パブリックまたはプライベートのオンラインイメージリポジトリに保存する必要があります。

> **Note**
>
> デプロイ時の `Dockerfile` を使用したカスタムイメージの構築は、Elastic Beanstalk 上の ECS マネージド Docker プラットフォームではサポートされていません。イメージを構築して、Elastic Beanstalk 環境を作成する前にオンラインレポジトリにデプロイします。

`Dockerrun.aws.json` v2 で、イメージを名前で指定します。

プライベートレポジトリを認証するように Elastic Beanstalk を設定するには、`Dockerrun.aws.json` v2 ファイルに `authentication` パラメータを含めます。

失敗したコンテナのデプロイ

Amazon ECS タスクが失敗した場合、Elastic Beanstalk 環境の 1 つ以上のコンテナが開始されません。Elastic Beanstalk は、Amazon ECS タスクが失敗したことで、マルチコンテナ環境をロールバックすることはありません。環境でコンテナの開始が失敗した場合は、Elastic Beanstalk コンソールから現在のバージョンまたは以前の機能するバージョンを再デプロイします。

既存のバージョンをデプロイするには

1. 環境のリージョンで Elastic Beanstalk コンソールを開きます。
2. アプリケーション名の右側の アクション をクリックし、アプリケーションバージョンの表示 をクリックします。
3. アプリケーションのバージョンを選択し、デプロイ をクリックします。

Elastic Beanstalk 用の ECS ベースの Docker プラットフォームの拡張

Elastic Beanstalk には、独自のコマンド、スクリプト、ソフトウェア、および設定をアプリケーションのデプロイに適用できる拡張機能が用意されています。ECS AL2 および AL2023 プラットフォームブランチのデプロイワークフローは、他の Linux ベースのプラットフォームとは若干異なります。詳細については、「Amazon Linux 2 以降で動作する ECS のインスタンスデプロイのワークフロー」を参照してください。

Elastic Beanstalk の ECS マネージド Docker 設定

この章では、ECS マネージド Docker 環境を設定する方法について説明します。次のリストは、この章で説明する設定項目をまとめたものです。

- `Dockerrun.aws.json` v2 – この設定ファイルは、イメージリポジトリと Docker イメージの名前などを指定します。
- EC2 インスタンスプロファイルロール – カスタムインスタンスプロファイルがある場合は、ECS がコンテナを管理するために必要なアクセス許可を最新の状態に保つように設定する方法を説明します。
- Elastic Load Balancing リスナー – デフォルトの HTTP ポートで実行されないプロキシやその他のサービスのインバウンドトラフィックをサポートする環境が必要な場合は、複数の Elastic Load Balancing リスナーを設定する必要があります。

トピック

- Dockerrun.aws.json v2 ファイルの設定
- コンテナ管理ポリシーと EC2 インスタンスロール
- 複数の Elastic Load Balancing リスナーの使用

Dockerrun.aws.json v2 ファイルの設定

`Dockerrun.aws.json v2` は、Elastic Beanstalk 環境の ECS クラスターでホストされる一連の Docker コンテナをデプロイする方法を記述する Elastic Beanstalk 設定ファイルです。Elastic Beanstalk プラットフォームは、ECS コンテナ定義を含む ECS タスク定義を作成します。これらの定義は `Dockerrun.aws.json` 設定ファイルに記述されます。

`Dockerrun.aws.json` ファイル内のコンテナ定義は、ECS クラスター内の各 Amazon EC2 インスタンスにデプロイするコンテナを記述します。この場合、Amazon EC2 インスタンスは Docker コンテナをホストするため、ホストコンテナインスタンスとも呼ばれます。設定ファイルは、Docker コンテナがマウントするホストコンテナインスタンス上に作成するデータボリュームも記述します。Elastic Beanstalk 上の ECS マネージド Docker 環境のコンポーネントの詳細と図については、この章の前半の「ECS マネージド Docker プラットフォームの概要」を参照してください。

`Dockerrun.aws.json` ファイルは単独で使用するか、1 つのアーカイブに追加のソースコードとともに圧縮できます。`Dockerrun.aws.json` でアーカイブされるソースコードは Amazon EC2 コンテナインスタンスにデプロイされ、`/var/app/current/` ディレクトリでアクセスできます。

トピック

- Dockerrun.aws.json v2
- ボリュームフォーマット

- 実行ロールの ARN 形式
- コンテナの定義形式
- 認証形式 – プライベートリポジトリからのイメージを使用
- Dockerrun.aws.json v2 の例

Dockerrun.aws.json v2

Dockerrun.aws.json ファイルには次のセクションが含まれています。

AWSEBDockerrunVersion

ECS マネージド Docker 環境のバージョン番号として値 2 を指定します。

executionRoleArn

アカウントに関連付けられたさまざまな目的とサービスのタスク実行 IAM ロールを指定します。アプリケーションがシークレットとして保存されている Elastic Beanstalk 環境変数を使用するには、必要なアクセス許可を付与するタスク実行ロールの ARN を指定する必要があります。その他の一般的なユースケースでも、このパラメータが必要になる場合があります。詳細については、「実行ロールの ARN 形式」を参照してください。

ボリューム

Amazon EC2 コンテナインスタンスのフォルダから、またはソースバンドル (/var/app/current にデプロイ) からボリュームを作成します。containerDefinitions セクションで mountPoints を使用している Docker コンテナ内のパスにこれらのボリュームをマウントします。

containerDefinitions

コンテナ定義の配列。

認証 (オプション)

プライベートリポジトリの認証データが含まれる .dockercfg ファイルの Amazon S3 内の場所です。

Dockerrun.aws.json の「containerDefinitions」と「volumes」のセクションは、Amazon ECS タスク定義ファイルの対応するセクションと同じ形式を使用します。タスク定義の形式およびタスク定義パラメータの完全な一覧については、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon ECS タスク定義」を参照してください。

ボリュームフォーマット

volume パラメータは、Amazon EC2 コンテナインスタンス内のフォルダ、またはソースバンドル (`/var/app/current` にデプロイされている) のいずれかからボリュームを作成します。

ボリュームは次の形式で指定します:

```
"volumes": [
    {
      "name": "volumename",
      "host": {
        "sourcePath": "/path/on/host/instance"
      }
    }
  ],
```

コンテナ定義の `mountPoints` を使用して、これらのボリュームを Docker コンテナ内のパスにマウントします。

Elastic Beanstalk は、コンテナごとにログ用の追加のボリュームを設定します。これらのボリュームは、ホストインスタンスにログを書き込むために、Docker コンテナによってマウントされる必要があります。

詳細については、次の「コンテナ定義の形式」セクションの「`mountPoints`」フィールドを参照してください。

実行ロールの ARN 形式

アプリケーションがシークレットとして保存されている Elastic Beanstalk 環境変数を使用するには、タスク実行 IAM ロールを指定する必要があります。ロールは、機密データを参照する AWS Secrets Manager シークレットまたは Parameter Store AWS Systems Manager パラメータを使用してユーザーに代わって AWS API コールを行うアクセス許可を Amazon ECS コンテナに付与する必要があります。アカウントに必要なアクセス許可を持つタスク実行 IAM ロールを作成する手順については、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon ECS タスク実行 IAM ロール」を参照してください。

```
{
"AWSEBDockerrunVersion": 2,
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
```

Amazon ECS マネージド Docker プラットフォームに必要な追加のアクセス許可

ECS **iam:PassRole**への EC2 インスタンスプロファイル許可

EC2 インスタンスプロファイルがこのロールを ECS コンテナに付与できるようにするには、次の例に示す`iam:PassRole`アクセス許可を含める必要があります。は、EC2 インスタンスにタスク実行ロールを ECS コンテナに渡すアクセス許可`iam:PassRole`を付与します。

この例では、ロールを ECS サービスにのみ渡すように EC2 インスタンスを制限します。この条件は必須ではありませんが、ベストプラクティスに従って共有アクセス許可の範囲を減らすために追加します。これを行うには、 `Condition`要素を使用します。

> **Note**
>
> ECS IAM タスク実行ロールを使用するには、 アクセス`iam:PassRole`許可が必要です。ECS タスク実行マネージドサービスロールを必要とする一般的なユースケースは他にもあります。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon ECS タスク実行 IAM ロール」を参照してください。

Example **iam:PassRole** アクセス許可を持つ ポリシー

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::123456789012:role/ecs-task-execution-role"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        }
    ]
```

```
}
```

シークレットとパラメータに Amazon ECS コンテナエージェントへのアクセス権を付与する

Amazon ECS タスク実行 IAM ロールには、シークレットとパラメータストアにアクセスするためのアクセス許可も必要です。EC2 インスタンスプロファイルロールの要件と同様に、ECS コンテナエージェントには、必要な Secrets Manager または Systems Manager リソースをプルするためのアクセス許可が必要です。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「Secrets Manager または Systems Manager のアクセス許可」を参照してください。

シークレットとパラメータに Elastic Beanstalk EC2 インスタンスへのアクセスを許可する

環境変数として設定されたシークレットをサポートするには、EC2 インスタンスプロファイルにアクセス許可を追加する必要があります。詳細については、「Elastic Beanstalk 環境変数へのシークレットとパラメータの取得」および「Secrets Manager に必要な IAM アクセス許可」を参照してください。

次の例では、前の`iam:PassRole`例と、参照先の で提供されている例を組み合わせていますSecrets Manager に必要な IAM アクセス許可。EC2 インスタンスが にアクセスするために必要なアクセス許可を追加 AWS Secrets Manager し、シークレット用に設定された Elastic Beanstalk 環境変数を初期化するためにシークレットとパラメータデータを取得 AWS Systems Manager するために を保存します。

Example アクセス**`iam:PassRole`**許可と Secrets Manager ポリシーの組み合わせ

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::123456789012:role/ecs-task-execution-role"
            ],
            "Condition": {
                "StringLike": {
```

```
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        },
        {
            "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"
            ]
        }
    ]
}
```

Example Systems Manager ポリシーと **iam:PassRole** アクセス許可の組み合わせ

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::123456789012:role/ecs-task-execution-role"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter",
                "kms:Decrypt"
```

```
            ],
            "Resource": [
                "arn:aws:ssm:us-east-1:111122223333:parameter/my-parameter",
                "arn:aws:kms:us-east-1:111122223333:key/my-key"
            ]
        }
    ]
}
```

コンテナの定義形式

次の例は、[containerDefinitions] セクションで一般的に使用されるパラメータのサブセットを示しています。そのほかのオプションパラメータも使用可能です。

Beanstalk プラットフォームは、ECS コンテナ定義を含む ECS タスク定義を作成します。Beanstalk は、ECS コンテナ定義のパラメータのサブセットをサポートします。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「コンテナ定義」を参照してください。

`Dockerrun.aws.json` ファイルには、次のフィールドとともに 1 つ以上のコンテナ定義オブジェクトの配列が含まれます。

name

コンテナの名前。最大長と使用できる文字については、「標準のコンテナ定義のパラメータ」を参照してください。

イメージ

Docker コンテナの構築元となるオンライン Docker リポジトリの Docker イメージの名前。次の規則があります。

- Docker ハブの公式リポジトリのイメージでは、1 つの名前 (例: `ubuntu`、`mongo`) を使用します。
- Docker ハブの他のリポジトリのイメージは、組織名で修飾されます (例: `amazon/amazon-ecs-agent`) 。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: `quay.io/assemblyline/ubuntu`)。

環境

コンテナに渡す環境変数の配列。

たとえば、次のエントリでは、**Container** という名前と **PHP** という値を使用して環境変数を定義しています。

```
"environment": [
  {
    "name": "Container",
    "value": "PHP"
  }
],
```

essential

コンテナが失敗した場合にタスクを停止する場合は True。重要でないコンテナは、インスタンスで他のコンテナに影響を与えることなく、終了またはクラッシュできます。

メモリ

コンテナ用に予約するコンテナインスタンスのメモリの量。コンテナ定義で `memory` と `memoryReservation` パラメータの一方または両方に 0 以外の整数を指定します。

memoryReservation

コンテナ用に予約するメモリのソフト制限 (MiB 単位)。コンテナ定義で `memory` と `memoryReservation` パラメータの一方または両方に 0 以外の整数を指定します。

mountPoints

マウントする Amazon EC2 コンテナインスタンスのボリュームと、それらをマウントする Docker コンテナファイルシステム上の場所。アプリケーションコンテンツを含むボリュームをマウントすると、コンテナはソースバンドルにアップロードしたデータを読み取ることができます。ログデータを書き込むためのログボリュームをマウントすると、Elastic Beanstalk は、これらのボリュームからログデータを収集することができます。

Elastic Beanstalk はコンテナインスタンスにログボリュームを作成します。Docker コンテナごとに 1 つ、場所は `/var/log/containers/`*`containername`* です。これらのボリュームの名前は `awseb-logs-`*`containername`* で、ログが書き込まれるコンテナファイル構造内の場所にマウントします。

たとえば、次のマウントポイントは、コンテナの nginx ログの場所を、`nginx-proxy` コンテナ用に Elastic Beanstalk が生成したボリュームにマッピングします。

```
{
  "sourceVolume": "awseb-logs-nginx-proxy",
```

```
  "containerPath": "/var/log/nginx"
}
```

portMappings

コンテナのネットワークポートをホストのポートにマッピングします。

links

リンク先のコンテナのリスト。リンクされたコンテナはお互いを検出し、安全に通信できます。

volumesFrom

別コンテナからのボリュームをすべてマウントします。たとえば、web という名前のコンテナからボリュームをマウントするには、次の手順を実行します。

```
"volumesFrom": [
  {
    "sourceContainer": "web"
  }
],
```

認証形式 – プライベートリポジトリからのイメージを使用

authentication セクションには、プライベートリポジトリの認証データが含まれています。このエントリはオプションです。

認証ファイルを authentication ファイルの Dockerrun.aws.json パラメータ内に含む Amazon S3 バケットに関する情報を追加します。authentication パラメータに有効な Amazon S3 バケットとキーが含まれていることを確認します。Amazon S3 バケットは、バケットを使用している環境と同じリージョンでホストする必要があります。Elastic Beanstalk は、他のリージョンでホストされている Amazon S3 バケットからファイルをダウンロードしません。

以下の形式が使用されます。

```
"authentication": {
    "bucket": "amzn-s3-demo-bucket",
    "key": "mydockercfg"
  },
```

認証ファイルの作成とアップロードについては、「イメージリポジトリを使用した認証」を参照してください。

Dockerrun.aws.json v2 の例

以下のスニペットは、2 つのコンテナを持つインスタンスの `Dockerrun.aws.json` ファイルの構文を示す例です。

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
        "sourcePath": "/var/app/current/php-app"
      }
    },
    {
      "name": "nginx-proxy-conf",
      "host": {
        "sourcePath": "/var/app/current/proxy/conf.d"
      }
    }
  ],
  "containerDefinitions": [
    {
      "name": "php-app",
      "image": "php:fpm",
      "environment": [
        {
          "name": "Container",
          "value": "PHP"
        }
      ],
      "essential": true,
      "memory": 128,
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "nginx-proxy",
      "image": "nginx",
```

```
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ],
      "links": [
        "php-app"
      ],
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        },
        {
          "sourceVolume": "nginx-proxy-conf",
          "containerPath": "/etc/nginx/conf.d",
          "readOnly": true
        },
        {
          "sourceVolume": "awseb-logs-nginx-proxy",
          "containerPath": "/var/log/nginx"
        }
      ]
    }
  ]
}
```

コンテナ管理ポリシーと EC2 インスタンスロール

Elastic Beanstalk コンソールで環境を作成すると、`AWSElasticBeanstalkMulticontainerDocker`管理ポリシーを含むデフォルトのインスタンスプロファイルを作成するように求められます。したがって、最初は、デフォルトの EC2 インスタンスプロファイルにこの管理ポリシーを含める必要があります。環境がデフォルトの代わりにカスタム EC2 インスタンスプロファイルロールを使用している場合`AWSElasticBeanstalkMulticontainerDocker`は、コンテナ管理に必要なアクセス許可をup-to-date状態に保つために、 管理ポリシーがアタッチされていることを確認してください。

Elastic Beanstalk は、Docker コンテナ内で実行される Amazon ECS コンテナエージェントを含んだ Amazon ECS 最適化 AMI を使用します。エージェントは、Amazon ECS と通信してコンテナのデプロイを調整します。Amazon ECS と通信するためには、各 Amazon EC2 インスタンスには対応する IAM アクセス許可が必要です。これらのアクセス許可は管理ポリシーで指定されます。このアクセス許可を表示するには、「AWS 管理ポリシーリファレンスガイド」の「AWSElasticBeanstalkMulticontainerDocker」を参照してください。

AWS Secrets Manager または Parameter Store に保存されているシークレットまたはパラメータにアクセスするように設定された Elastic Beanstalk AWS Systems Manager 環境変数を使用する場合は、追加のアクセス許可で EC2 インスタンスプロファイルをカスタマイズする必要があります。詳細については、「実行ロールの ARN 形式」を参照してください。

複数の Elastic Load Balancing リスナーの使用

デフォルトの HTTP ポートで実行されないプロキシまたは他のサービス用の受信トラフィックをサポートするため、ECS マネージド Docker 環境で、複数の Elastic Load Balancing リスナーを設定できます。

ソースバンドルで `.ebextensions` フォルダを作成し、ファイル拡張子が `.config` のファイルを追加します。次の例では、ポート 8080 で Elastic Load Balancing リスナーを作成する設定ファイルを示します。

`.ebextensions/elb-listener.config`

```
option_settings:
  aws:elb:listener:8080:
    ListenerProtocol: HTTP
    InstanceProtocol: HTTP
    InstancePort: 8080
```

作成したカスタム Amazon Virtual Private Cloud (Amazon VPC) が環境で実行されている場合、残りはElastic Beanstalk が処理します。デフォルトの VPC では、インスタンスのセキュリティグループを設定して、ロードバランサーからの着信を許可する必要があります。進入ルールを追加する第 2 の設定ファイルをセキュリティグループに追加します。

`.ebextensions/elb-ingress.config`

```
Resources:
  port8080SecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
```

```
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 8080
      FromPort: 8080
      SourceSecurityGroupName: { "Fn::GetAtt": ["AWSEBLoadBalancer",
 "SourceSecurityGroup.GroupName"] }
```

設定ファイルの形式の詳細については、「Elastic Beanstalk 環境リソースの追加とカスタマイズ」および「オプション設定」を参照してください。

Elastic Load Balancing 設定にリスナーを追加し、セキュリティグループでポートを開くことに加えて、`Dockerrun.aws.json` v2 ファイルの `containerDefinitions` セクションで、Docker コンテナのポートにホストインスタンスのポートをマッピングする必要があります。例を以下に示します。

```
"portMappings": [
  {
    "hostPort": 8080,
    "containerPort": 8080
  }
]
```

`Dockerrun.aws.json` v2 ファイル形式の詳細については、「`Dockerrun.aws.json` v2」を参照してください。

Elastic Beanstalk コンソールを使用した ECS マネージド Docker 環境の作成

このチュートリアルでは、2 つのコンテナを使用する ECS マネージド Docker 環境用のコンテナの設定およびソースコードの準備について説明します。

コンテナ、PHP アプリケーション、および nginx プロキシは、Elastic Beanstalk 環境の各 Amazon Elastic Compute Cloud (Amazon EC2) インスタンスで並列に実行されます。環境を作成し、アプリケーションが実行中であることを確認したら、コンテナインスタンスに接続して、それらの状態を確認できます。

セクション

- ECS マネージド Docker コンテナの定義
- コンテンツの追加

- Elastic Beanstalk にデプロイする
- コンテナインスタンスへの接続
- Amazon ECS コンテナエージェントを検査する

ECS マネージド Docker コンテナの定義

新しい Docker 環境作成の最初のステップは、アプリケーションデータ用のディレクトリの作成です。このフォルダはローカルマシンの任意の場所に配置でき、任意の名前を付けることができます。コンテナ設定ファイルに加えて、このフォルダには、Elastic Beanstalk にアップロードして環境にデプロイするコンテンツが含まれます。

> **Note**
>
> このチュートリアルのすべてのコードは、GitHub (https://github.com/awslabs/eb-docker-nginx-proxy) の awslabs レポジトリで入手できます。

Amazon EC2 インスタンスのコンテナの設定に Elastic Beanstalk が使用するファイルは、`Dockerrun.aws.json` v2 という名前の JSON 形式のテキストファイルです。ECS マネージド Docker プラットフォームバージョンは、このファイルのバージョン 2 形式を使用します。この形式は、ECS 管理の Docker プラットフォームでのみ使用できます。ECS によって管理されていない Docker プラットフォームブランチをサポートする他の設定ファイルバージョンとは大きく異なるためです。

アプリケーションのルートで `Dockerrun.aws.json` v2 テキストファイルを作成し、次のテキストを追加します。

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
        "sourcePath": "/var/app/current/php-app"
      }
    },
    {
      "name": "nginx-proxy-conf",
      "host": {
```

```
        "sourcePath": "/var/app/current/proxy/conf.d"
      }
    }
  ],
  "containerDefinitions": [
    {
      "name": "php-app",
      "image": "php:fpm",
      "essential": true,
      "memory": 128,
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "nginx-proxy",
      "image": "nginx",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ],
      "links": [
        "php-app"
      ],
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        },
        {
          "sourceVolume": "nginx-proxy-conf",
          "containerPath": "/etc/nginx/conf.d",
          "readOnly": true
        },
        {
```

```
          "sourceVolume": "awseb-logs-nginx-proxy",
          "containerPath": "/var/log/nginx"
        }
      ]
    }
  ]
}
```

この例の設定では、2 つのコンテナ、PHP ウェブサイトとその前面の nginx プロキシを定義します。これらの 2 つコンテナは Elastic Beanstalk 環境の各インスタンスの Docker コンテナで並列に実行され、このファイルで定義されるホストインスタンスのボリュームの共有コンテンツ (ウェブサイトのコンテンツ) にアクセスします。コンテナそのものは、Docker ハブの公式リポジトリでホストされているイメージから作成されます。環境は、次のようになります。

設定で定義されるボリュームは、アプリケーションソースバンドルの一部として次に作成し、アップロードするコンテンツに対応します。コンテナは、コンテナ定義の `mountPoints` セクションでボリュームをマウントすることで、ホストのコンテンツにアクセスします。

`Dockerrun.aws.json` v2 の形式とそのパラメータの詳細については、「コンテナの定義形式」を参照してください。

コンテンツの追加

次に、PHP サイトで閲覧者に表示するコンテンツと、nginx プロキシ用の設定ファイルを追加します。

php-app/index.php

```
<h1>Hello World!!!</h1>
<h3>PHP Version <pre><?= phpversion()?></pre></h3>
```

php-app/static.html

```
<h1>Hello World!</h1>
<h3>This is a static HTML page.</h3>
```

proxy/conf.d/default.conf

```
server {
```

```
  listen 80;
  server_name localhost;
  root /var/www/html;

  index index.php;

  location ~ [^/]\.php(/|$) {
    fastcgi_split_path_info ^(.+?\.php)(/.*)$;
    if (!-f $document_root$fastcgi_script_name) {
      return 404;
    }

    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;

    fastcgi_pass php-app:9000;
    fastcgi_index index.php;
  }
}
```

Elastic Beanstalk にデプロイする

アプリケーションフォルダーには以下のファイルが含まれます。

```
### Dockerrun.aws.json
### php-app
#   ### index.php
#   ### static.html
### proxy
    ### conf.d
        ### default.conf
```

Elastic Beanstalk 環境を作成するために必要なのは、これだけです。上記のファイルとフォルダの `.zip` アーカイブを作成します (最上位プロジェクトフォルダを含みません)。Windows エクスプローラーでアーカイブを作成するには、プロジェクトフォルダの内容を選択し、右クリックして 送る を選択し、圧縮 (zip 形式) フォルダ をクリックします。

> **Note**
>
> 必要なファイル構造の詳細、および他の環境でアーカイブを作成する手順については、「」を参照してください[Elastic Beanstalk アプリケーションソースバンドルを作成する]

次に、ソースバンドルを Elastic Beanstalk にアップロードして、環境を作成します。[Platform (プラットフォーム)] で [Docker] を選択します。[プラットフォームブランチ] では、[64 ビット版 Amazon Linux 2023 で動作する ECS] を選択します。

環境を起動するには (コンソール)

1. この事前に設定されたリンク: console.aws.amazon.com/elasticBeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced を使用して、Elastic Beanstalk コンソールを開きます。
2. [プラットフォーム] で、アプリケーションで使用される言語に一致するプラットフォームとプラットフォームブランチか、コンテナベースアプリケーション用の Docker プラットフォームを選択します。
3. [アプリケーションコード] で、[コードのアップロード] を選択します。
4. ローカルファイル を選択し、[Choose file (ファイルの選択)] を選択して、ソースバンドルを開きます。
5. 確認と起動 を選択します。
6. 使用できる設定を確認し、アプリの作成 を選択します。

Elastic Beanstalk コンソールにより、新しい環境の管理ダッシュボードにリダイレクトされます。この画面には、環境の状態ステータスと、Elastic Beanstalk サービスによって出力されたイベントが表示されます。状態が緑色になったら、環境名の横の URL をクリックして新しいウェブサイトを表示します。

コンテナインスタンスへの接続

次に、Elastic Beanstalk 環境で Amazon EC2 インスタンスに接続し、いくつかの可動部分を動かしてみます。

環境内のインスタンスに接続するための最も簡単な方法は、EB CLI を使用することです。これを使用するには、EB CLI をインストールします (まだ行っていない場合)。また、Amazon EC2 SSH キーペアを使用して環境を設定する必要があります。コンソールの セキュリティ設定ページまたは EB

CLI の eb init コマンドを使用して、この操作を行います。環境のインスタンスに接続するには、EB CLI の eb ssh コマンドを使用します。

これで、Docker コンテナをホストする Amazon EC2 インスタンスに接続したので、設定を確認できます。`ls` で `/var/app/current` を実行します。

```
[ec2-user@ip-10-0-0-117 ~]$ ls /var/app/current
Dockerrun.aws.json  php-app  proxy
```

このディレクトリには、環境の作成中に Elastic Beanstalk にアップロードしたソースバンドルからのファイルが含まれます。

```
[ec2-user@ip-10-0-0-117 ~]$ ls /var/log/containers
nginx-proxy    nginx-proxy-4ba868dbb7f3-stdouterr.log
php-app        php-app-dcc3b3c8522c-stdouterr.log       rotated
```

ここでは、コンテナインスタンスにログが作成され、Elastic Beanstalk によって収集されます。Elastic Beanstalk は、各コンテナ用にこのディレクトリでボリュームを作成します。このボリュームは、ログが書き込まれるコンテナの場所にマウントします。

Docker を確認し、`docker ps` で実行中のコンテナを表示することもできます。

```
[ec2-user@ip-10-0-0-117 ~]$ sudo docker ps
CONTAINER ID   IMAGE                              COMMAND                  CREATED
  STATUS                  PORTS                                NAMES

4ba868dbb7f3   nginx                              "/docker-entrypoint.…"   4 minutes ago
   Up 4 minutes            0.0.0.0:80->80/tcp, :::80->80/tcp   ecs-awseb-Tutorials-env-
dc2aywfjwg-1-nginx-proxy-acca84ef87c4aca15400
dcc3b3c8522c   php:fpm                            "docker-php-entrypoi…"   4 minutes ago
   Up 4 minutes            9000/tcp                             ecs-awseb-Tutorials-env-
dc2aywfjwg-1-php-app-b8d38ae288b7b09e8101
d9367c0baad6   amazon/amazon-ecs-agent:latest     "/agent"                 5 minutes ago
  Up 5 minutes (healthy)                                        ecs-agent
```

デプロイした 2 つの実行中のコンテナと、デプロイを調整した Amazon ECS コンテナエージェントが表示されます。

Amazon ECS コンテナエージェントを検査する

Elastic Beanstalk 上の ECS マネージド Docker 環境の Amazon EC2 インスタンスは、Docker コンテナでエージェントプロセスを実行します。このエージェントは、コンテナのデプロイを調整するために、Amazon ECS サービスに接続します。これらのデプロイは Amazon ECS でタスクとして実行され、タスク定義ファイルで設定されます。Elastic Beanstalk は、ソースバンドルでアップロードする `Dockerrun.aws.json` に基づいて、これらのタスク定義ファイルを作成します。

`http://localhost:51678/v1/metadata` への HTTP GET リクエストで、コンテナエージェントの状態を確認します。

```
[ec2-user@ip-10-0-0-117 ~]$ curl http://localhost:51678/v1/metadata
{
  "Cluster":"awseb-Tutorials-env-dc2aywfjwg",
  "ContainerInstanceArn":"arn:aws:ecs:us-west-2:123456789012:container-instance/awseb-
Tutorials-env-dc2aywfjwg/db7be5215cd74658aacfcb292a6b944f",
  "Version":"Amazon ECS Agent - v1.57.1 (089b7b64)"
}
```

この構造は、Amazon ECS クラスター名、およびクラスターインスタンス (接続先の Amazon EC2 インスタンス) の ARN (Amazon リソースネーム) を示します。

詳細については、`http://localhost:51678/v1/tasks` への HTTP GET リクエストを行ってください。

```
[ec2-user@ip-10-0-0-117 ~]$ curl http://localhost:51678/v1/tasks
{
   "Tasks":[
      {
         "Arn":"arn:aws:ecs:us-west-2:123456789012:task/awseb-Tutorials-env-dc2aywfjwg/
bbde7ebe1d4e4537ab1336340150a6d6",
         "DesiredStatus":"RUNNING",
         "KnownStatus":"RUNNING",
         "Family":"awseb-Tutorials-env-dc2aywfjwg",
         "Version":"1",
         "Containers":[
            {

 "DockerId":"dcc3b3c8522cb9510b7359689163814c0f1453b36b237204a3fd7a0b445d2ea6",
               "DockerName":"ecs-awseb-Tutorials-env-dc2aywfjwg-1-php-app-
b8d38ae288b7b09e8101",
               "Name":"php-app",
```

```
                "Volumes":[
                   {
                      "Source":"/var/app/current/php-app",
                      "Destination":"/var/www/html"
                   }
                ]
             },
             {

 "DockerId":"4ba868dbb7f3fb3328b8afeb2cb6cf03e3cb1cdd5b109e470f767d50b2c3e303",
                "DockerName":"ecs-awseb-Tutorials-env-dc2aywfjwg-1-nginx-proxy-
acca84ef87c4aca15400",
                "Name":"nginx-proxy",
                "Ports":[
                   {
                      "ContainerPort":80,
                      "Protocol":"tcp",
                      "HostPort":80
                   },
                   {
                      "ContainerPort":80,
                      "Protocol":"tcp",
                      "HostPort":80
                   }
                ],
                "Volumes":[
                   {
                      "Source":"/var/app/current/php-app",
                      "Destination":"/var/www/html"
                   },
                   {
                      "Source":"/var/log/containers/nginx-proxy",
                      "Destination":"/var/log/nginx"
                   },
                   {
                      "Source":"/var/app/current/proxy/conf.d",
                      "Destination":"/etc/nginx/conf.d"
                   }
                ]
             }
          ]
       }
    ]
```

```
}
```

この構造は、このチュートリアルのサンプルプロジェクトから 2 つの Docker コンテナをデプロイするために実行されるタスクについて示しています。以下の情報が表示されます。

- [KnownStatus] – `RUNNING` ステータスは、コンテナがまだアクティブであることを示します。
- ファミリー – Elastic Beanstalk が `Dockerrun.aws.json` から作成したタスク定義の名前。
- [Version] – タスク定義のバージョン。これは、タスク定義ファイルを更新するたびに増えていきます。
- [Containers] – インスタンスで実行されるコンテナに関する情報。

さらに多くの情報が、Amazon ECS サービス自体から使用できます。このサービスは を使用して呼び出すことができます AWS Command Line Interface Amazon ECS AWS CLI で を使用する手順、および Amazon ECS 全般については、「Amazon ECS ユーザーガイド」を参照してください。

Elastic Beanstalk アプリケーションを AL1 の ECS マネージドマルチコンテナ Docker から Amazon Linux 2023 の ECS に移行する

> **Note**
>
> 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。

このトピックでは、廃止されるプラットフォームブランチである 64 ビット版 Amazon Linux 上で実行されるマルチコンテナ Docker から 64 ビット版 AL2023 上で実行される ECS にアプリケーションを移行する方法について説明します。このターゲットプラットフォームブランチは最新であり、サポートされています。以前のマルチコンテナ Docker AL1 ブランチと同様に、新しい ECS AL2023 プラットフォームブランチは Amazon ECS を使用して、複数の Docker コンテナを Elastic Beanstalk 環境内の Amazon ECS クラスターにデプロイするように調整します。新しい ECS AL2023 プラットフォームブランチでは、以前のマルチコンテナ Docker AL1 プラットフォームブランチのすべての機能がサポートされています。また、同じ `Dockerrun.aws.json` v2 ファイルがサポートされています。

セクション

- Elastic Beanstalk コンソールを使用した移行

- を使用して移行する AWS CLI

Elastic Beanstalk コンソールを使用した移行

Elastic Beanstalk コンソールを使用して移行するには、同じソースコードを AL2023 上で動作する ECS プラットフォームブランチをベースにした新しい環境にデプロイします。ソースコードを変更する必要はありません。

Amazon Linux 2023 上で動作する ECS プラットフォームブランチに移行するには

1. 古い環境に既にデプロイされているアプリケーションソースを使用して、アプリケーションソースバンドルを作成します。同じアプリケーションソースバンドル、および同じ `Dockerrun.aws.json` v2 ファイルを使用できます。
2. Amazon Linux 2023 上で動作する ECS プラットフォームブランチを使用して、新しい環境を作成します。アプリケーションコードの前のステップからのソースバンドルを使用します。詳細なステップについては、この章で前述されている「ECS マネージド Docker チュートリアル」の「Elastic Beanstalk にデプロイする」を参照してください。

を使用して移行する AWS CLI

また、 AWS Command Line Interface （ AWS CLI) を使用して、既存のマルチコンテナ Docker Amazon Linux Docker 環境を新しい ECS AL2023 プラットフォームブランチに移行することもできます。この場合、新しい環境を作成したり、ソースコードを再デプロイする必要はありません。実行する必要があるのは AWS CLI 、update-environment コマンドのみです。このコマンドにより、プラットフォームの更新が実行され、既存の環境が ECS Amazon Linux 2023 プラットフォームブランチに移行されます。

以下の構文を使用して、環境を新しいプラットフォームブランチに移行します。

```
aws elasticbeanstalk update-environment \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 version running ECS" \
--region my-region
```

以下は、環境 beta-101 を us-east-1 リージョンにある ECS Amazon Linux 2023 プラットフォームブランチのバージョン 3.0.0 に移行するコマンドの例です。

```
aws elasticbeanstalk update-environment \
```

```
--environment-name beta-101 \
--solution-stack-name "64bit Amazon Linux 2023 v4.0.0 running ECS" \
--region us-east-1
```

solution-stack-name パラメータは、プラットフォームブランチとそのバージョンを提供します。適切なソリューションスタック名を指定して、最新のプラットフォームブランチのバージョンを使用します。上の例に示すように、すべてのプラットフォームブランチのバージョンがソリューションスタック名に含まれています。Docker プラットフォームの最新のソリューションスタックのリストについては、「AWS Elastic Beanstalk プラットフォーム」ガイドの「サポートされているプラットフォーム」を参照してください。

> **Note**
>
> list-available-solution-stacks コマンドは、 AWS リージョンのアカウントで使用できるプラットフォームバージョンのリストを提供します。
>
> ```
> aws elasticbeanstalk list-available-solution-stacks --region us-east-1 --query
> SolutionStacks
> ```

の詳細については AWS CLI、 AWS Command Line Interface ユーザーガイドを参照してください。Elastic Beanstalk の AWS CLI コマンドの詳細については、AWS CLI 「Elastic Beanstalk のコマンドリファレンス」を参照してください。

イメージリポジトリを使用した認証

このトピックでは、Elastic Beanstalk を使用してオンラインイメージリポジトリを認証する方法について説明します。プライベートリポジトリの場合、Elastic Beanstalk はイメージをプルしてデプロイする前に認証する必要があります。Amazon ECR Public の場合、認証はオプションですが、レート制限が高く、信頼性が向上します。

Amazon ECR リポジトリからのイメージを使用する

Amazon Elastic Container Registry (Amazon ECR) AWS を使用して、カスタム Docker イメージをに保存できます。

Docker イメージを Amazon ECR に保存すると、Elastic Beanstalk は環境のインスタンスプロファイルを使用して Amazon ECR レジストリに対して自動的に認証します。したがって、Amazon ECR リポジトリ内のイメージにアクセスするためのアクセス許可をインスタンスに提供する必要があり

ます。これを行うには、AmazonEC2ContainerRegistryReadOnly 管理ポリシーをインスタンスプロファイルにアタッチして、環境のインスタンスプロファイルにアクセス許可を追加します。これにより、アカウント内のすべての Amazon ECR リポジトリへの読み取り専用アクセスが可能になります。また、次のテンプレートを使用してカスタムポリシーを作成することで、単一のリポジトリにのみアクセスすることもできます。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEbAuth",
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "AllowPull",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:ecr:us-east-2:111122223333:repository/repository-name"
            ],
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:BatchGetImage"
            ]
        }
    ]
}
```

上記のポリシーの Amazon リソースネーム（ARN）をリポジトリの ARN に置き換えます。

Dockerrun.aws.json ファイル内のイメージ情報を指定する必要があります。設定は、使用するプラットフォームによって異なります。

ECS マネージド Docker プラットフォームでは、次のようにコンテナ定義オブジェクト で image キーを使用します。

```
"containerDefinitions": [
        {
        "name": "my-image",
        "image": "account-id.dkr.ecr.us-east-2.amazonaws.com/repository-name:latest",
```

Docker プラットフォームについては、URL でイメージを参照してください。次のように、URL は Dockerrun.aws.json ファイルの Image 定義に含まれます。

```
  "Image": {
      "Name": "account-id.dkr.ecr.us-east-2.amazonaws.com/repository-name:latest",
      "Update": "true"
    },
```

AWS Systems Manager (SSM) パラメータストアまたは の使用 AWS Secrets Manager

デプロイ前にプライベートリポジトリで認証するように Elastic Beanstalk を設定し、コンテナイメージへのアクセスを有効にします。

このアプローチでは、2 つのコンポーネントを含む Elastic Beanstalk デプロイプロセスのビルド前フェーズを使用します。

- リポジトリ認証情報を保存する環境変数を定義する ebextensions
- イメージをプルdocker loginする前に実行するプラットフォームフックスクリプト

フックスクリプトは、 AWS Systems Manager Parameter Store または から入力された環境変数から認証情報を安全に取得します AWS Secrets Manager。この機能には、2025 年 3 月 26 日以降にリリースされた Elastic Beanstalk Docker および ECS マネージド Docker プラットフォームが必要です。詳細については、「環境変数の設定」を参照してください。

Parameter Store または を使用してプライベートリポジトリに対して認証するように Elastic Beanstalk AWS Systems Manager を設定するには AWS Secrets Manager

> **Note**
>
> 続行する前に、Parameter Store または AWS Systems Manager AWS Secrets Manager で認証情報を設定し、必要な IAM アクセス許可を設定してください。詳細については、「シークレットを環境変数として設定するための前提条件」を参照してください。

1. プロジェクト用に次のディレクトリ構造を作成します。

```
### .ebextensions
#   ### env.config
### .platform
#   ### confighooks
#   #   ### prebuild
#   #       ### 01login.sh
#   ### hooks
#       ### prebuild
#           ### 01login.sh
### Dockerfile
```

2. Parameter Store AWS Systems Manager または AWS Secrets Managerを使用して、プライベートリポジトリの認証情報を保存します。この例では、Parameter Store AWS Systems Manager と AWS Secrets Manager の両方を示していますが、これらのサービスの 1 つだけを使用することを選択できます。

```
aws ssm put-parameter --name USER --type SecureString --value "username"
aws secretsmanager create-secret --name PASSWD --secret-string "passwd"
```

3. 次の `env.config` ファイルを作成し、前述のディレクトリ構造に示すように、`.ebextensions` ディレクトリに配置します。この設定では、aws:elasticbeanstalk:application:environmentsecrets名前空間を使用して、 `USER`および `PASSWD` Elastic Beanstalk 環境変数を Systems Manager パラメータストアに保存されている値に初期化します。

> **Note**
>
> 変数名 USERと が put-parameter コマンドと create-secret コマンドで使用されるパラメータ名PASSWDと一致することを確認します。

```
option_settings:
  aws:elasticbeanstalk:application:environmentsecrets:
    USER: arn:aws:ssm:us-east-1:111122223333:parameter/user
    PASSWD: arn:aws:secretsmanager:us-east-1:111122223333:passwd
```

4. 次の `01login.sh` スクリプトファイルを作成し、次のディレクトリに配置します (前述のディレクトリ構造にも示されています)。

   - `.platform/confighooks/prebuild`
   - `.platform/hooks/prebuild`

```
### example 01login.sh
#!/bin/bash
echo $PASSWD | docker login -u $USER --password-stdin
```

   この`01login.sh`スクリプトは、ステップ 3 で設定された環境変数を使用し、 docker loginを介してパスワードを に安全に渡しますstdin。Docker 認証の詳細については、Docker ドキュメントの「Docker ログイン」を参照してください。

> **メモ**
>
> - フックファイルは、バイナリファイル、またはインタプリタパスを含む #! 行で始まるスクリプトファイル (#!/bin/bash など) です。
> - 詳細については、「Elastic Beanstalk Linux プラットフォームの拡張」の「the section called “プラットフォームフック”」を参照してください。

認証が設定されると、Elastic Beanstalk はプライベートリポジトリからイメージをプルしてデプロイできます。

Dockerrun.aws.json ファイルの使用

このセクションでは、プライベートリポジトリに対して Elastic Beanstalk を認証する別の方法について説明します。この方法では、Docker コマンドを使用して認証ファイルを生成し、認証ファイルを Amazon S3 バケットにアップロードします。また、`Dockerrun.aws.json` ファイルにバケット情報を含める必要もあります。

認証ファイルを生成して Elastic Beanstalk に提供するには

1. docker login コマンドを使用して認証ファイルを生成します。Docker Hub のリポジトリでは、docker login を実行します。

   ```
   $ docker login
   ```

   他のレジストリでは、レジストリサーバーの URL を入力します。

   ```
   $ docker login registry-server-url
   ```

   > **Note**
   >
   > Elastic Beanstalk 環境で (Amazon Linux 2 より前の) Amazon Linux AMI Docker プラットフォームバージョンを使用している場合は、「the section called “(Amazon Linux 2 より前の) Amazon Linux AMI での Docker 設定”」の関連情報をお読みください。

   認証ファイルの詳細については、Docker ウェブサイトのDocker ハブにイメージを保存するおよびdocker ログインを参照してください。

2. `.dockercfg` という名前の認証ファイルのコピーを安全な Amazon S3 バケットにアップロードします。

   - Amazon S3 バケットは、使用している環境 AWS リージョン と同じ でホストされている必要があります。Elastic Beanstalk は、他のリージョンでホストされている Amazon S3 バケットからファイルをダウンロードすることはできません。
   - インスタンスプロファイル内の IAM ロールに `s3:GetObject` オペレーションを許可します。詳細については、「」を参照してくださいElastic Beanstalk インスタンスプロファイルの管理

3. Amazon S3 バケット情報を、Authentication ファイルの Dockerrun.aws.json パラメータに含めます。

   次の例は、サードパーティーレジストリでプライベートイメージを使用するように、mydockercfg というバケットに amzn-s3-demo-bucket という認証ファイルを使用する方法を示しています。AWSEBDockerrunVersion の正しいバージョン番号については、例の後の注を参照してください。

```
{
  "AWSEBDockerrunVersion": "version-no",
  "Authentication": {
    "Bucket": "amzn-s3-demo-bucket",
    "Key": "mydockercfg"
  },
  "Image": {
    "Name": "quay.io/johndoe/private-image",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": "1234"
    }
  ],
  "Volumes": [
    {
      "HostDirectory": "/var/app/mydb",
      "ContainerDirectory": "/etc/mysql"
    }
  ],
  "Logging": "/var/log/nginx"
}
```

Dockerrun.aws.json バージョン

AWSEBDockerrunVersion パラメータは、Dockerrun.aws.json ファイルのバージョンを示します。

- Docker AL2 および AL2023 プラットフォームでは、次のバージョンのファイルを使用します。
  - Dockerrun.aws.json v3 – Docker Compose を使用する環境。

- `Dockerrun.aws.json v1` – Docker Compose を使用しない環境。
- Amazon Linux 2 上で実行される ECS と AL2023 上で実行される ECS は `Dockerrun.aws.json v2` ファイルを使用します。廃止されたプラットフォームである ECS - マルチコンテナ Docker Amazon Linux AMI (AL1) も同じバージョンを使用していました。

Elastic Beanstalk がプライベートリポジトリをホストするオンラインレジストリで認証した後、イメージをデプロイしてプルできます。

Amazon ECR Public からのイメージの使用

Amazon ECR Public は、Docker イメージをホストするパブリックコンテナレジストリです。Amazon ECR Public リポジトリはパブリックにアクセス可能ですが、認証によりデプロイのレート制限が高くなり、信頼性が向上します。

> **Note**
>
> Amazon ECR パブリック認証は、中国リージョン (`cn-*`) および AWS GovCloud リージョン () ではサポートされていませんus-gov-*。これらのリージョンでは、Elastic Beanstalk は認証されていないプルを使用します。

Amazon ECR パブリック認証を有効にするには、環境のインスタンスプロファイルに次のアクセス許可を追加します。Amazon ECR パブリック認証の詳細については、「Amazon Elastic Container Registry Public User Guide」の「Registry authentication in Amazon ECR public」を参照してください。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
          "Effect": "Allow",
          "Action": [
             "ecr-public:GetAuthorizationToken",
             "sts:GetServiceBearerToken"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

これらのアクセス許可がインスタンスプロファイルにアタッチされると、Elastic Beanstalk は Amazon ECR Public レジストリで自動的に認証します。Amazon ECR Public イメージは、`Dockerrun.aws.json` ファイルまたは Dockerfile の標準`public.ecr.aws/`*`registry-alias`*`/`*`repository-name:tag`*形式を使用して参照できます。

Elastic Beanstalk Docker 環境の設定

この章では、ECS マネージド Docker プラットフォームブランチなど、サポートされているすべての Docker プラットフォームブランチのその他の設定情報について説明します。特定のプラットフォームブランチまたはプラットフォームブランチコンポーネントがセクションで特定されていない限り、サポートされている Docker および ECS 管理の Docker プラットフォームを実行しているすべての環境に適用されます。

> **Note**
>
> Elastic Beanstalk 環境で (Amazon Linux 2 より前の) Amazon Linux AMI Docker プラットフォームバージョンを使用している場合は、「the section called “(Amazon Linux 2 より前の) Amazon Linux AMI での Docker 設定”」の追加情報を必ずお読みください。

セクション

- Docker 環境でのソフトウェアの設定
- コンテナ内の環境変数の参照
- Docker Compose を使用した環境変数の補間機能の使用
- Docker Compose を使用した拡張ヘルスレポート用のログの生成
- Docker Compose を使用した Docker コンテナのカスタマイズされたログ記録
- Docker イメージ
- Docker 環境に対する管理された更新の設定
- Docker 設定の名前空間

- (Amazon Linux 2 より前の) Amazon Linux AMI での Docker 設定

Docker 環境でのソフトウェアの設定

Elastic Beanstalk コンソールを使用して、お客様の環境のインスタンスで実行しているソフトウェアを設定できます。

Elastic Beanstalk コンソールで Docker 環境を設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. 必要な設定変更を行います。
6. ページの最下部で [適用] を選択し変更を保存します。

任意の環境でソフトウェア設定を定義する方法については、「the section called “環境変数とソフトウェア設定”」を参照してください。以下のセクションでは、Docker 固有の情報を取り上げます。

コンテナオプション

[コンテナオプション] セクションには、プラットフォーム固有のオプションがあります。Docker 環境では、環境に NGINX プロキシサーバーを含めるかどうかを選択できます。

Docker Compose ありの環境

Docker Compose ありの Docker 環境を管理する場合、Elastic Beanstalk はプロキシサーバーをコンテナとして実行すると想定します。したがって、プロキシサーバー設定のデフォルトは [なし] に設定され、Elastic Beanstalk は NGINX 設定を提供しません。

> **Note**
>
> プロキシサーバーとして NGINX を選択しても、Docker Compose ありの環境ではこの設定は無視されます。プロキシサーバーのデフォルト設定は [なし] のままです。

Docker Compose ありの Amazon Linux 2 プラットフォーム上の Docker では、NGINX ウェブサーバープロキシが無効になっているため、拡張ヘルスレポート用のログを生成するための手順に従う

必要があります。詳細については、「Docker Compose を使用した拡張ヘルスレポート用のログの生成」を参照してください。

環境プロパティ (環境変数)

環境プロパティ (環境変数とも呼ばれます) を使用して、エンドポイント、デバッグ設定、その他の情報などの値をアプリケーションに渡すことができます。コンソールの環境変数セクションでは、アプリケーションを実行している EC2 インスタンスで環境変数を指定できます。環境変数は、キーと値のペアとしてアプリケーションに渡されます。

コンテナで実行されるアプリケーションコードは、名前で環境変数を参照し、その値を読み取ることができます。これらの環境変数を読み取るソースコードは、プログラミング言語によって異なります。Elastic Beanstalk マネージドプラットフォームがサポートするプログラミング言語での環境変数値を読み取る手順は、それぞれのプラットフォームのトピックで見つかります。これらのトピックへのリンクのリストについては、「the section called “環境変数とソフトウェア設定”」を参照してください。

Elastic Beanstalk 環境変数のシークレットとパラメータ

Elastic Beanstalk は、環境変数で AWS Secrets Manager および AWS Systems Manager Parameter Store データを参照する機能を提供します。これは、アプリケーションが API コールを管理することなく、これらのサービスに保存されているシークレットとパラメータにネイティブにアクセスする安全なオプションです。この機能をサポートするには、Elastic Beanstalk Docker および ECS マネージド Docker プラットフォームが 2025 年 3 月 26 日以降にリリースされたバージョンである必要があります。環境変数を使用してシークレットを参照する方法の詳細については、「」を参照してくださいElastic Beanstalk 環境変数へのシークレットとパラメータの取得。

Docker Compose ありの環境

Docker Compose ありの Docker 環境を管理する場合、コンテナ内の環境変数を取得するための追加設定を行う必要があります。コンテナで実行されている実行可能ファイルがこれらの環境変数にアクセスするには、`docker-compose.yml` でそれらを参照する必要があります。詳細については、「コンテナ内の環境変数の参照」を参照してください。

コンテナ内の環境変数の参照

Amazon Linux 2 Docker プラットフォームで Docker Compose ツールを使用している場合、アプリケーションプロジェクトのルートディレクトリに `.env` と呼ばれる Docker Compose 環境ファイルが Elastic Beanstalk により生成されます。このファイルには、Elastic Beanstalk に設定した環境変数が保存されます。

> **Note**
>
> アプリケーションバンドルに `.env` ファイルを含めると、Elastic Beanstalk は `.env` ファイルを生成しません。

Elastic Beanstalk で定義した環境変数をコンテナで参照するには、これらの設定方法のいずれかまたは両方に従う必要があります。

- Elastic Beanstalk によって生成された `.env` ファイルを、`env_file` ファイルの `docker-compose.yml` 設定オプションに追加します。
- `docker-compose.yml` ファイルで環境変数を直接定義します。

次のファイルはその例を示しています。サンプル `docker-compose.yml` ファイルは、両方のアプローチを示しています。

- 環境プロパティ `DEBUG_LEVEL=1` および `LOG_LEVEL=error` を定義すると、Elastic Beanstalk によって次の `.env` ファイルが自動的に生成されます。

```
DEBUG_LEVEL=1
LOG_LEVEL=error
```

- この `docker-compose.yml` ファイルでは、`env_file` 設定オプションは `.env` ファイルを指し、`DEBUG=1` ファイル内で環境変数 `docker-compose.yml` を直接定義します。

```
services:
  web:
    build: .
    environment:
      - DEBUG=1
    env_file:
      - .env
```

> **メモ**
>
> - 両方のファイルで同じ環境変数を設定した場合、`docker-compose.yml` ファイルで定義された変数の優先順位は、`.env` ファイルで定義された変数よりも高くなります。

- 文字列にスペースが追加されないように、等号 (=) と変数に割り当てられた値の間にスペースを残さないように注意してください。

Docker Compose の環境変数について詳しくは、「Environment variables in Compose」を参照してください

Docker Compose を使用した環境変数の補間機能の使用

2023 年 7 月 28 日のプラットフォームリリース以降、Docker Amazon Linux 2 プラットフォームブランチには Docker Compose 補間機能が提供されます。この機能により、Compose ファイルの値を変数で設定し、ランタイムに補間することができます。補間機能の詳細については、Docker ドキュメントウェブサイトの「補間」を参照してください。

> **Important**
>
> この機能をアプリケーションで使用する場合は、プラットフォームフックを使用するアプローチを実装する必要があることに注意してください。
> これが必要なのは、プラットフォームエンジンに実装した緩和策があるためです。この緩和策により、新しい補間機能を知らないお客様や、環境変数を $ 文字を用いて利用している既存のアプリケーションに対しても、後方互換性を確保することができます。更新されたプラットフォームエンジンは、デフォルトで $ 文字を $$ 文字に置き換えることで補間をエスケープします。

次は、補間機能を使用できるように設定できるプラットフォームフックスクリプトの例です。

```
#!/bin/bash

: '
example data format in .env file
key1=value1
key2=value2
'
envfile="/var/app/staging/.env"
tempfile=$(mktemp)

while IFS= read -r line; do
  # split each env var string at '='
```

```
  split_str=(${line//=/ })
  if [ ${#split_str[@]} -eq 2 ]; then
    # replace '$$' with '$'
    replaced_str=${split_str[1]//\$\$/\$}
    # update the value of env var using ${replaced_str}
    line="${split_str[0]}=${replaced_str}"
  fi
  # append the updated env var to the tempfile
  echo "${line}" #"${tempfile}"
done < "${envfile}"
# replace the original .env file with the tempfile
mv "${tempfile}" "${envfile}"
```

プラットフォームフックを次の両方のディレクトリに配置します。

- `.platform/confighooks/predeploy/`
- `.platform/hooks/predeploy/`

詳細については、このガイドの「Linux プラットフォームの拡張」トピックにある プラットフォームフック を参照してください。

Docker Compose を使用した拡張ヘルスレポート用のログの生成

Elastic Beanstalk ヘルスエージェントは、Elastic Beanstalk 環境のオペレーティングシステムとアプリケーションのヘルスメトリクスを提供します。これは、特定の形式で情報を中継するウェブサーバーのログ形式に依存します。

Elastic Beanstalk は、ウェブサーバープロキシをコンテナとして実行することを前提としています。その結果、Docker Composeを実行している Docker 環境では、NGINX ウェブサーバープロキシが無効になります。サーバーを構成して、Elastic Beanstalk ヘルスエージェントが使用する場所と形式でログを書き込む必要があります。これにより、ウェブサーバープロキシが無効になっていても、拡張ヘルスレポートを最大限に活用できます。

これを行う手順については、「ウェブサーバーログ設定」を参照してください。

Docker Compose を使用した Docker コンテナのカスタマイズされたログ記録

問題を効率的にトラブルシューティングし、コンテナ化されたサービスをモニタリングするには、環境マネジメントコンソールまたは EB CLI を通じて Elastic Beanstalk からインスタンスログをリクエ

ストします。インスタンスログは、バンドルログとテールログで構成され、組み合わされてパッケージ化されるため、ログと最近のイベントを効率的かつわかりやすい方法で表示できます。

Elastic Beanstalk は、コンテナインスタンス上の `docker-compose.yml` にログディレクトリを作成します (`/var/log/eb-docker/containers/<service name>` ファイルで定義されたサービスごとに1 つずつ)。Amazon Linux 2 Docker プラットフォームで Docker Compose 機能を使用している場合は、ログが書き込まれるコンテナファイル構造内の場所にこれらのディレクトリをマウントできます。ログデータを書き込むためのログディレクトリをマウントすると、Elastic Beanstalk はこれらのディレクトリからログデータを収集することができます。

テールファイルとバンドルログを取得するようにサービスのログファイルを設定するには

1. `docker-compose.yml` ファイルを編集します。
2. サービスの `volumes` キーの下に、次のようにバインドマウントを追加します。

   `"${EB_LOG_BASE_DIR}/<service name>:<log directory inside container>`

   次のサンプル `docker-compose.yml` ファイルで、以下の操作を行います。

   - `nginx-proxy` は `<#####>` です
   - `/var/log/nginx` は `<##############>`

   ```
   services:
     nginx-proxy:
       image: "nginx"
       volumes:
         - "${EB_LOG_BASE_DIR}/nginx-proxy:/var/log/nginx"
   ```

- `var/log/nginx` ディレクトリには、コンテナ内の nginx-proxy サービスのログが含まれており、ホスト上の `/var/log/eb-docker/containers/nginx-proxy` ディレクトリにマップされます。
- このディレクトリ内のすべてのログが、Elastic Beanstalk のリクエストインスタンスログ機能を通じてバンドルログおよびテールログとして取得できるようになりました。

> **メモ**
>
> - ${EB_LOG_BASE_DIR} は、値 `/var/log/eb-docker/containers` を使用して Elastic Beanstalk により設定された環境変数です。
> - Elastic Beanstalk は、`/var/log/eb-docker/containers/<service name>` ファイル内の各サービスの `docker-compose.yml` ディレクトリを自動的に作成します。

Docker イメージ

Elastic Beanstalk 用の Docker および ECS マネージドの Docker プラットフォームブランチでは、パブリック/プライベートのオンラインイメージリポジトリに保存された Docker イメージの使用がサポートされています。

`Dockerrun.aws.json` で名前によってイメージを指定します。次の規則があります。

- Docker ハブの公式リポジトリのイメージでは、1 つの名前 (例: `ubuntu`、`mongo`) を使用します。
- Docker ハブの他のリポジトリのイメージは、組織名で修飾されます (例: `amazon/amazon-ecs-agent`)。
- 他のオンラインリポジトリのイメージは、さらにドメイン名で修飾されます (例: `quay.io/assemblyline/ubuntu` または *`account-id`*`.dkr.ecr.us-east-2.amazonaws.com/ubuntu:trusty`)。

Docker プラットフォームのみを使用する環境では、Dockerfile を使用して環境の作成中に独自のイメージを作成することもできます。詳細については、「Dockerfile を使用したカスタムイメージの構築」を参照してください。ECS マネージド Docker プラットフォームはこの機能をサポートしていません。

Docker 環境に対する管理された更新の設定

管理されたプラットフォームの更新により、スケジュールに基づいて、環境を自動的に最新バージョンのプラットフォームに更新するように設定できます。

Docker 環境の場合、新しいプラットフォームバージョンに新しい Docker バージョンが含まれるときに、Docker バージョン間でプラットフォームの自動更新を実行するかどうか決定できます。2.9.0 以降の Docker プラットフォームバージョンを実行している環境から更新する場

合、Elastic Beanstalk は、Docker バージョン間のマネージドプラットフォーム更新をサポートします。新しいプラットフォームバージョンに新しいバージョンの Docker が含まれている場合、Elastic Beanstalk はマイナーバージョン番号を増分します。したがって、Docker バージョン間で管理されたプラットフォームの更新を許可するには、マイナーおよびパッチバージョンの両方の更新について、管理されたプラットフォームの更新を有効にします。Docker バージョン間で管理されたプラットフォームの更新が行われないようにする場合は、パッチバージョンの更新のみを適用するように、管理されたプラットフォームの更新を有効にします。

たとえば、次の設定ファイルは、マイナーおよびパッチバージョンの両方の更新で、毎週火曜日の午前 9:00 UTC に管理されたプラットフォームの更新を有効にし、Docker バージョン間の管理された更新が行われるようにします。

Example .ebextensions/managed-platform-update.config

```
option_settings:
  aws:elasticbeanstalk:managedactions:
    ManagedActionsEnabled: true
    PreferredStartTime: "Tue:09:00"
  aws:elasticbeanstalk:managedactions:platformupdate:
    UpdateLevel: minor
```

Docker バージョン 2.9.0 以前を実行している環境では、新しいプラットフォームバージョンに新しい Docker バージョンが含まれている場合、Elastic Beanstalk がマネージドプラットフォームの更新を実行することはありません。

Docker 設定の名前空間

設定ファイルを使用して、設定オプションを設定し、デプロイの間、他のインスタンス設定タスクを実行できます。設定オプションは、プラットフォーム固有のものでも、Elastic Beanstalk サービス全体のすべてのプラットフォームに適用できるものでもかまいません。設定オプションは、名前空間として整理されています。

> **Note**
>
> この情報は、Docker Compose を実行していない Docker 環境にのみ適用されます。このオプションは、Docker Compose を実行する Docker 環境では動作が異なります。Docker Compose のあるプロキシサービスの詳細については、「コンテナオプション」を参照してください。

Docker プラットフォームでは、すべての Elastic Beanstalk 環境でサポートされるオプションに加えて、以下の名前空間でサポートされるオプションがあります。

- `aws:elasticbeanstalk:environment:proxy` – 環境のプロキシサーバーを選択します。Docker では、Nginx の実行ありまたはプロキシサーバーの実行なしがサポートされています。

以下の設定ファイルの例では、プロキシサーバーを実行しないように Docker 環境を設定しています。

Example .ebextensions/docker-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: none
```

(Amazon Linux 2 より前の) Amazon Linux AMI での Docker 設定

Elastic Beanstalk Docker 環境で (Amazon Linux 2 より前の) Amazon Linux AMI プラットフォームバージョンを使用している場合は、このセクションの追加情報を読んでください。

プライベートリポジトリの認証ファイルの使用

この情報は、プライベートリポジトリからイメージを使用している場合に該当します。Docker のバージョン 1.7 以降で、docker login コマンドによって作成される認証ファイルの名前と形式が変更されました。(Amazon Linux 2 より前の) Amazon Linux AMI Docker プラットフォームバージョンには、古い `~/.dockercfg` 形式の設定ファイルが必要です。

Docker バージョン 1.7 以降では、docker login コマンドによって `~/.docker/config.json` に以下の形式の認証ファイルが作成されます。

```
{
    "auths":{
      "server":{
        "auth":"key"
      }
    }
  }
```

Docker バージョン 1.6.2 以前では、docker login コマンドは `~/.dockercfg` に以下の形式の認証ファイルを作成します。

```
{
    "server" :
    {
      "auth" : "auth_token",
      "email" : "email"
    }
  }
```

`config.json` ファイルを変換するには、外側の `auths` キーを削除し、`email` キーを追加し、古い形式と一致するように JSON ドキュメントをフラット化します。

Amazon Linux 2 Docker プラットフォームバージョンでは、Elastic Beanstalk によって新しい認証ファイル名と形式が使用されます。Amazon Linux 2 Docker プラットフォームバージョンを使用している場合は、docker login コマンドによって作成される認証ファイルを変換せずに使用できます。

追加ストレージボリュームの設定

Amazon Linux AMI のパフォーマンスを向上させるために、Elastic Beanstalk は、Docker 環境の Amazon EC2 インスタンス用に 2 つの Amazon EBS ストレージボリュームを設定します。すべての Elastic Beanstalk 環境用にプロビジョニングされたルートボリュームに加え、`xvdcz` という名前の 2 つ目の 12 GB ボリュームが、Docker 環境のイメージ保管用にプロビジョニングされます。

Docker イメージ用にさらなるストレージスペースや IOPS が必要な場合は、イメージストレージボリュームを aws:autoscaling:launchconfiguration 名前空間の `BlockDeviceMapping` 設定オプションを使用してカスタマイズできます。

たとえば、次の設定ファイルは、ストレージボリュームサイズを 500 プロビジョンド IOPS を持つ 100 GB に増加します。

Example .ebextensions/blockdevice-xvdcz.config

```
option_settings:
  aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: /dev/xvdcz=:100::io1:500
```

アプリケーションの追加ボリュームの設定に `BlockDeviceMappings` オプションを使用する場合、確実に作成するために `xvdcz` のためのマッピングを含める必要があります。次の例では、デフォル

ト設定のイメージストレージボリューム xvdcz および追加の sdh という名前の 24 GB アプリケーションボリュームという 2 つのボリュームを設定しています。

Example .ebextensions/blockdevice-sdh.config

```
option_settings:
  aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: /dev/xvdcz=:12:true:gp2,/dev/sdh=:24
```

> **Note**
>
> この名前空間の設定を変更する場合、Elastic Beanstalk は環境のすべてのインスタンスを新しい構成で実行しているインスタンスに置き換えます。詳細については、「設定変更」を参照してください。

レガシープラットフォーム

この章では、 でサポートされなくなった以前の Docker プラットフォームに関連するコンテンツを一覧表示します AWS Elastic Beanstalk。ここにリストされているトピックは、廃止前にこれらの機能やコンポーネントを使用していたお客様向けのリファレンスとして、このドキュメントに残ります。

トピック

- Amazon Linux 上で動作するマルチコンテナ Docker から Amazon Linux 2 で動作する Elastic Beanstalk Docker への移行
- Elastic Beanstalk で事前設定された Docker GlassFish コンテナ

Amazon Linux 上で動作するマルチコンテナ Docker から Amazon Linux 2 で動作する Elastic Beanstalk Docker への移行

64 ビット版 Amazon Linux 2 上で動作する ECS プラットフォームブランチのリリースに先立って、Elastic Beanstalk では、64 ビット版 Amazon Linux 上で動作するマルチコンテナ Docker プラットフォームブランチをベースにした環境を使用するお客様に Amazon Linux 2 への代替移行パスが提供されています。このトピックでは、その移行パスについて説明します。このトピックは、移行パスを完了したすべてのお客様の参照用ドキュメントとして使用できます。

64 ビット版 Amazon Linux 上で動作するマルチコンテナ Docker プラットフォームブランチをベースにした環境を使用するお客様は、64 ビット版 Amazon Linux 2 上で動作する ECS プラットフォームブランチに移行することをお勧めします。代替移行パスとは異なり、このアプローチでは引き続き Amazon ECS を使用して、ECS マネージド Docker 環境へのコンテナのデプロイを調整します。これにより、より直接的なアプローチが可能になります。ソースコードに変更を加える必要はなく、同じ `Dockerrun.aws.json` v2 がサポートされています。詳細については、「Elastic Beanstalk アプリケーションを AL1 の ECS マネージドマルチコンテナ Docker から Amazon Linux 2023 の ECS に移行する」を参照してください。

Amazon Linux 上のマルチコンテナ Docker から Docker Amazon Linux 2 プラットフォームブランチへのレガシー移行

Amazon Linux AMI のマルチコンテナ Docker プラットフォームで実行されているアプリケーションを Amazon Linux 2 Docker プラットフォームに移行できます。Amazon Linux AMI のマルチコンテナ Docker プラットフォームでは、コンテナとして実行するためにビルド済みのアプリケーションイメージを指定する必要があります。移行後は、Amazon Linux 2 Docker プラットフォームでも Elastic Beanstalk がデプロイ中にコンテナイメージを構築できるため、この制限はなくなります。アプリケーションはマルチコンテナ環境で引き続き実行され、Docker Compose ツールの利点が追加されます。

Docker Compose は、マルチコンテナ Docker アプリケーションを定義および実行するためのツールです。Docker Compose の詳細およびインストール方法については、Docker サイトの「Docker Compose の概要」および「Docker Compose のインストール」を参照してください。

`docker-compose.yml` ファイル

Docker Compose ツールは、アプリケーションサービスの設定に `docker-compose.yml` ファイルを使用します。このファイルは、アプリケーションプロジェクトディレクトリおよびアプリケーションソースバンドル内の `Dockerrun.aws.json v2` ファイルを置き換えます。`docker-compose.yml` ファイルを手動で作成すると、ほとんどのパラメータ値について `Dockerrun.aws.json v2` ファイルを参照すると便利です。

以下は、同じアプリケーションの `docker-compose.yml` ファイルと対応する `Dockerrun.aws.json v2` ファイルの例です。`docker-compose.yml` ファイルの詳細については、「Compose file reference」を参照してください。`Dockerrun.aws.json v2` ファイルの詳細については、「`Dockerrun.aws.json v2`」を参照してください。

docker-compose.yml

```
version: '2.4'
services:
  php-app:
    image: "php:fpm"
    volumes:
      - "./php-app:/var/www/html:ro
"
      - "${EB_LOG_BASE_DIR}/php-app
:/var/log/sample-app"
    mem_limit: 128m
    environment:
      Container: PHP
  nginx-proxy:
    image: "nginx"
    ports:
      - "80:80"
    volumes:
      - "./php-app:/var/www/html:ro
"
      - "./proxy/conf.d:/etc/nginx/
conf.d:ro"
      - "${EB_LOG_BASE_DIR}/nginx-p
roxy:/var/log/nginx"
    mem_limit: 128m
    links:
      - php-app
```

Dockerrun.aws.json v2

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
        "sourcePath": "/var/app/
current/php-app"
      }
    },
    {
      "name": "nginx-proxy-conf",
      "host": {
        "sourcePath": "/var/app/
current/proxy/conf.d"
      }
    }
  ],
  "containerDefinitions": [
    {
      "name": "php-app",
      "image": "php:fpm",
      "environment": [
        {
          "name": "Container",
          "value": "PHP"
        }
      ],
      "essential": true,
      "memory": 128,
      "mountPoints": [
        {
          "sourceVolume": "php-app"
,
          "containerPath": "/var/www
/html",
          "readOnly": true
        }
      ]
    },
```

docker-compose.yml	Dockerrun.aws.json v2

```
    {
      "name": "nginx-proxy",
      "image": "nginx",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ],
      "links": [
        "php-app"
      ],
      "mountPoints": [
        {
          "sourceVolume": "php-app"
,
          "containerPath": "/var/www
/html",
          "readOnly": true
        },
        {
          "sourceVolume": "nginx-pr
oxy-conf",
          "containerPath": "/etc/ngi
nx/conf.d",
          "readOnly": true
        },
        {
          "sourceVolume": "awseb-lo
gs-nginx-proxy",
          "containerPath": "/var/log
/nginx"
        }
      ]
    }
  ]
}
```

移行に関するその他の考慮事項

Docker Amazon Linux 2 プラットフォームとマルチコンテナ Docker Amazon Linux AMI プラットフォームでは、環境プロパティの実装方法が異なります。これら 2 つのプラットフォームには、Elastic Beanstalk がコンテナごとに作成する異なるログディレクトリもあります。Amazon Linux AMI マルチコンテナー Docker プラットフォームから移行した後、新しい Amazon Linux 2 Docker プラットフォーム環境におけるこれらの異なる実装に注意する必要があります。

エリア	Docker Compose を使用した Amazon Linux 2 の Docker プラットフォーム	Amazon Linux AMI のマルチコンテナ Docker プラットフォーム
環境プロパティ	コンテナが環境プロパティにアクセスするには、`.env` ファイル内の `docker-compose.yml` ファイルへの参照を追加する必要があります。Elastic Beanstalk は `.env` ファイルを生成し、各プロパティを環境変数としてリスト化します。詳細については、「コンテナ内の環境変数の参照」を参照してください。	Elastic Beanstalk は環境プロパティをコンテナに直接渡すことができます。コンテナ内で実行されているコードは、追加の設定なしに、環境変数としてこれらのプロパティにアクセスできます。
ログディレクトリ	各コンテナについて、Elastic Beanstalk は `/var/log/eb-docker/containers/` *`<service name>`* (または `${EB_LOG_BASE_DIR}/<service name>`)というログディレクトリを作成します。詳細については、「Docker Compose を使用した Docker コンテナのカスタマイズされたログ記録」を参照してください。	各コンテナについて、Elastic Beanstalk は `/var/log/containers/` *`<containername>`* というログディレクトリを作成します。詳細については、「コンテナの定義形式」の `mountPoints` フィールドを参照してください。

移行手順

Amazon Linux 2 Docker プラットフォームに移行するには

1. 既存の `Dockerrun.aws.json v2` ファイルに基づいて、アプリケーション用の `docker-compose.yml` ファイルを作成します。詳細については、前述の「docker-compose.yml ファイル」セクションを参照してください。
2. アプリケーションプロジェクトフォルダのルートディレクトリで、`Dockerrun.aws.json v2` ファイルを先ほど作成した `docker-compose.yml` に置き換えます。

   ディレクトリ構造は以下のようになります。

   ```
   ~/myApplication
   |-- docker-compose.yml
   |-- .ebextensions
   |-- php-app
   |-- proxy
   ```

3. eb init コマンドを使用して、Elastic Beanstalk にデプロイするローカルディレクトリを設定します。

   ```
   ~/myApplication$ eb init -p docker application-name
   ```

4. eb create コマンドを使用して、環境を作成し、Docker イメージをデプロイします。

   ```
   ~/myApplication$ eb create environment-name
   ```

5. アプリがウェブアプリケーションの場合、環境の起動後に eb open コマンドを使用してウェブブラウザに表示します。

   ```
   ~/myApplication$ eb open environment-name
   ```

6. eb status コマンドを使用して、新しく作成した環境のステータスを表示できます。

   ```
   ~/myApplication$ eb status environment-name
   ```

Elastic Beanstalk で事前設定された Docker GlassFish コンテナ

> **Note**
>
> 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

Amazon Linux AMI (AL1) で動作する、事前設定済みの Docker GlassFish プラットフォームブランチは、サポートされなくなりました。GlassFish アプリケーションをサポートされている Amazon Linux 2023 に移行するには、GlassFish とアプリケーションコードを Amazon Linux 2023 Docker イメージにデプロイします。詳細については、次の「the section called “チュートリアル - Docker での GlassFish: Amazon Linux 2023 へのパス”」トピックを参照してください。

事前設定済みの Docker コンテナの使用開始 - (Amazon Linux 2 より前の) Amazon Linux AMI で

このセクションでは、事前設定された Docker コンテナを使用して、ローカルでアプリケーション例を開発し、そのアプリケーションを Elastic Beanstalk にデプロイする方法について説明します。

ローカルの開発環境のセットアップ

このチュートリアルでは、GlassFish アプリケーション例を使用します。

使用する環境をセットアップするには

1. アプリケーション例用の新しいフォルダーを作成します。

```
~$ mkdir eb-preconf-example
~$ cd eb-preconf-example
```

2. アプリケーション例のコードを新しいフォルダにダウンロードします。

```
~$ wget https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/samples/docker-
glassfish-v1.zip
~$ unzip docker-glassfish-v1.zip
~$ rm docker-glassfish-v1.zip
```

ローカル環境での開発とテスト

GlassFish アプリケーション例を開発するには

1. `Dockerfile` をアプリケーションのルートフォルダに追加します。ファイルで、ローカルの事前設定済み AWS Elastic Beanstalk Docker コンテナの実行に使用する Docker ベースイメージを指定します。後で、アプリケーションを Elastic Beanstalk の事前設定された Docker GlassFish プラットフォームバージョンにデプロイします。このプラットフォームバージョンが使用する Docker ベースイメージを選択します。プラットフォームバージョンの現在の Docker イメージを検出するには、「AWS Elastic Beanstalk プラットフォーム」ガイドの「AWS Elastic Beanstalk でサポートされるプラットフォーム」ページにある「事前設定済み Docker コンテナ」セクションを参照してください。

   Example ~/Eb-preconf-example/Dockerfile

   ```
   # For Glassfish 5.0 Java 8
   FROM amazon/aws-eb-glassfish:5.0-al-onbuild-2.11.1
   ```

   `Dockerfile` の使用方法の詳細については、「Elastic Beanstalk へのデプロイ用に Docker イメージを準備する」を参照してください。

2. Docker イメージを作成します。

   ```
   ~/eb-preconf-example$ docker build -t my-app-image .
   ```

3. イメージから Docker コンテナを実行します。

   > **Note**
   >
   > コンテナのポート 8080 をローカルホストのポート 3000 にマッピングする `-p` フラグを指定する必要があります。Elastic Beanstalk Docker コンテナでは、常にコンテナのポート 8080 でアプリケーションを公開します。`-it` フラグは、イメージをインタラクティブプロセスとして実行します。`--rm` フラグは、コンテナが存在する場合にコンテナファイルシステムをクリーンアップします。オプションで、イメージをデーモンとして実行するための `-d` フラグも指定できます。

   ```
   $ docker run -it --rm -p 3000:8080 my-app-image
   ```

4. アプリケーション例を表示するには、次の URL をウェブブラウザに打ち込みます。

```
http://localhost:3000
```

Elastic Beanstalk にデプロイします

アプリケーションをテストすると、Elastic Beanstalk にデプロイする準備が完了します。

アプリケーションを Elastic Beanstalk にデプロイするには

1. アプリケーションのルートフォルダで、`Dockerfile` を `Dockerfile.local` に名前変更します。このステップは、Elastic Beanstalk に対する正しい指示を含んだ `Dockerfile` を Elastic Beanstalk で使用し、カスタマイズされた Docker イメージを Elastic Beanstalk 環境の各 Amazon EC2 インスタンスに作成するために必要なステップです。

   > **Note**
   > プラットフォームバージョンのベースの Docker イメージを変更する命令が `Dockerfile` に含まれている場合、このステップを行う必要はありません。`Dockerfile` に、コンテナの構築に使用するベースイメージを指定する `Dockerfile` 行のみが含まれている場合、`FROM` を使用する必要は一切ありません。この場合、その `Dockerfile` は重複しています。

2. アプリケーションソースバンドルを作成します。

```
~/eb-preconf-example$ zip myapp.zip -r *
```

3. 事前に設定されたリンク: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced を使用して、Elastic Beanstalk コンソールを開きます。
4. [Platform (プラットフォーム)] の [Preconfigured – Docker (事前設定 Docker)] で、[Glassfish] を選択します。
5. [Application code] では、[Upload your code] を選択して、[Upload] を選択します。
6. [Local file (ローカルファイル)]、[Browse (参照)] の順に選択したら、先ほど作成したアプリケーションソースバンドルを開きます。
7. [アップロード] を選択します。

8. 確認と起動 を選択します。
9. 使用できる設定を確認し、アプリの作成 を選択します。
10. 環境が作成されると、デプロイされたアプリケーションを表示できます。コンソールダッシュボードの上部に表示される環境 URL を選択します。

Docker プラットフォームへの GlassFish アプリケーションのデプロイ: Amazon Linux 2023 への移行パス

このチュートリアルの目的は、事前設定された Docker GlassFish プラットフォーム (Amazon Linux AMI に基づく) を使用しているお客様に、Amazon Linux 2023 への移行パスを示すことです。GlassFish アプリケーションを Amazon Linux 2023 に移行するには、GlassFish とアプリケーションコードを Amazon Linux 2023 Docker イメージにデプロイします。

このチュートリアルでは、Docker AWS Elastic Beanstalk プラットフォームを使用して、Java EE GlassFish アプリケーションサーバーに基づくアプリケーションを Elastic Beanstalk 環境にデプロイする方法について説明します。

Docker イメージを構築する 2 つの方法を示します。

- シンプル – GlassFish アプリケーションのソースコードを提供し、Elastic Beanstalk 環境のプロビジョニングの一環として Docker イメージを構築して実行します。この方法は、設定が簡単ですが、インスタンスのプロビジョニング時間が長くなります。
- アドバンスト – アプリケーションコードと依存関係を含むカスタム Docker イメージを作成し、Elastic Beanstalk に提供して、お客様の環境で使用します。この方法は少し複雑ですが、お客様の環境でのインスタンスのプロビジョニング時間が短くなります。

前提条件

このチュートリアルでは、Elastic Beanstalk の基本的なオペレーション、Elastic Beanstalk コマンドラインコマンドラインインターフェイス (EB CLI) および Docker についてある程度の知識があることを前提としています。まだ起動していない場合は、Elastic Beanstalk の使用を開始する方法について説明します。 の指示に従って、最初の Elastic Beanstalk 環境を起動します。このチュートリアルでは EB CLI を使用しますが、Elastic Beanstalk コンソールを使用して環境を作成し、アプリケーションをアップロードすることもできます。

このチュートリアルに従うには、以下の Docker コンポーネントも必要です。

- Docker の稼働中のローカルインストール。詳細については、Docker ドキュメントウェブサイトの「Get Docker」を参照してください。
- Docker Hub へのアクセス。Docker ハブにアクセスするには、Docker ID を作成する必要があります。詳細については、Docker ドキュメントウェブサイトの「Share the application」を参照してください。

Elastic Beanstalk プラットフォームでの Docker 環境の設定の詳細については、この同じ章の「Elastic Beanstalk へのデプロイ用に Docker イメージを準備する」を参照してください。

シンプルの例: アプリケーションコードを提供する

これは、GlassFish アプリケーションをデプロイする簡単な方法です。このチュートリアルに含まれる `Dockerfile` と共に、アプリケーションのソースコードを用意します。Elastic Beanstalk は、アプリケーションと GlassFish ソフトウェアスタックを含む Docker イメージを構築します。その後、Elastic Beanstalk は環境インスタンスでイメージを実行します。

この方法の問題は、Elastic Beanstalk が環境のインスタンスを作成するたびに Docker イメージをローカルに構築することです。イメージの構築により、インスタンスのプロビジョニング時間が長くなります。この影響は、初期環境の作成に限らず、スケールアウトアクション中にも発生します。

GlassFish アプリケーション例で環境を起動するには

1. アプリケーション例 `docker-glassfish-al2-v1.zip` をダウンロードし、`.zip` ファイルを開発環境のディレクトリに展開します。

```
~$ curl https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/samples/docker-
glassfish-al2-v1.zip --output docker-glassfish-al2-v1.zip
~$ mkdir glassfish-example
~$ cd glassfish-example
~/glassfish-example$ unzip ../docker-glassfish-al2-v1.zip
```

ディレクトリ構造は以下のようになります。

```
~/glassfish-example
|-- Dockerfile
|-- Dockerrun.aws.json
|-- glassfish-start.sh
|-- index.jsp
|-- META-INF
|   |-- LICENSE.txt
```

```
|   |-- MANIFEST.MF
|   `-- NOTICE.txt
|-- robots.txt
`-- WEB-INF
    `-- web.xml
```

以下のファイルは、お客様の環境で Docker コンテナを構築して実行するために重要です。

- `Dockerfile` – アプリケーションと必要な依存関係を含むイメージを構築するために Docker が使用する手順を提供します。
- `glassfish-start.sh` – アプリケーションを起動するために Docker イメージが実行するシェルスクリプト。
- `Dockerrun.aws.json` – GlassFish アプリケーションサーバーのログをログファイルリクエストに含めるためのログ記録キーを提供します。GlassFish ログに関心がない場合は、このファイルを省略できます。

2. Elastic Beanstalk にデプロイするローカルディレクトリを設定します。

```
~/glassfish-example$ eb init -p docker glassfish-example
```

3. (オプション) eb local run コマンドを使用して、コンテナを構築し、ローカルで実行します。

```
~/glassfish-example$ eb local run --port 8080
```

> **Note**
>
> eb local コマンドの詳細については、「the section called “eb local”」を参照してください。このコマンドは Windows ではサポートされていません。または、docker build コマンドと docker run コマンドを使用してコンテナを構築して実行することもできます。詳細については、「Docker ドキュメント」を参照してください。

4. (オプション) コンテナが実行しているときに、eb local open コマンドを使用して、ウェブブラウザでアプリケーションを表示します。または、ウェブブラウザで http://localhost:8080/ を開きます。

```
~/glassfish-example$ eb local open
```

5. eb create コマンドを使用して、環境を作成し、アプリケーションをデプロイします。

```
~/glassfish-example$ eb create glassfish-example-env
```

6. 環境が起動したら、eb open コマンドを使用してウェブブラウザで表示します。

```
~/glassfish-example$ eb open
```

アプリケーション例を使用し終えたら、環境を終了し、関連するリソースを削除します。

```
~/glassfish-example$ eb terminate --all
```

アドバンストの例: 事前に構築された Docker イメージを提供する

これは、GlassFish アプリケーションをデプロイするためのより高度な方法です。最初の例に基づいて、アプリケーションコードと GlassFish ソフトウェアスタックを含む Docker イメージを作成し、そのイメージを Docker Hub にプッシュします。この 1 回限りのステップを完了したら、カスタムイメージに基づいて Elastic Beanstalk 環境を起動できます。

環境を起動して Docker イメージを提供すると、環境内のインスタンスはこのイメージを直接ダウンロードして使用するため、Docker イメージを構築する必要はありません。したがって、インスタンスのプロビジョニング時間が短くなります。

> **メモ**
>
> - 以下のステップでは、一般利用可能な Docker イメージを作成します。
> - ローカルの Docker インストールから Docker コマンドと、Docker Hub の資格情報を使用します。詳細については、このトピック内で前述した「前提条件」セクションを参照してください。

事前に構築された GlassFish アプリケーションの Docker イメージで環境を起動するには

1. 前のシンプルの例と同様に、アプリケーション例 `docker-glassfish-al2-v1.zip` をダウンロードして展開します。その例を完了済みの場合は、既存のディレクトリを使用できます。
2. Docker イメージを構築し、Docker Hub にプッシュします。*docker-id* に、あなたの Docker ID を入力して Docker Hub にサインインします。

```
~/glassfish-example$ docker build -t docker-id/beanstalk-glassfish-example:latest .
```

```
~/glassfish-example$ docker push docker-id/beanstalk-glassfish-example:latest
```

> **Note**
>
> イメージをプッシュする前に、docker login を実行しなければならないことがあります。パラメータなしでコマンドを実行すると、Docker Hub の資格情報の入力を求められます。

3. 追加のディレクトリを作成します。

```
~$ mkdir glassfish-prebuilt
~$ cd glassfish-prebuilt
```

4. 以下の例を `Dockerrun.aws.json` という名前のファイルにコピーします。

   Example **`~/glassfish-prebuilt/Dockerrun.aws.json`**

```
{
  "AWSEBDockerrunVersion": "1",
  "Image": {
    "Name": "docker-username/beanstalk-glassfish-example"
  },
  "Ports": [
    {
      "ContainerPort": 8080,
      "HostPort": 8080
    }
  ],
  "Logging": "/usr/local/glassfish5/glassfish/domains/domain1/logs"
}
```

5. Elastic Beanstalk にデプロイするローカルディレクトリを設定します。

```
~/glassfish-prebuilt$ eb init -p docker glassfish-prebuilt$
```

6. (オプション) eb local run コマンドを使用して、コンテナをローカルで実行します。

```
~/glassfish-prebuilt$ eb local run --port 8080
```

7. (オプション) コンテナが実行しているときに、eb local open コマンドを使用して、ウェブブラウザでアプリケーションを表示します。または、ウェブブラウザで http://localhost:8080/ を開きます。

```
~/glassfish-prebuilt$ eb local open
```

8. eb create コマンドを使用して、環境を作成し、Docker イメージをデプロイします。

```
~/glassfish-prebuilt$ eb create glassfish-prebuilt-env
```

9. 環境が起動したら、eb open コマンドを使用してウェブブラウザで表示します。

```
~/glassfish-prebuilt$ eb open
```

アプリケーション例を使用し終えたら、環境を終了し、関連するリソースを削除します。

```
~/glassfish-prebuilt$ eb terminate --all
```

Elastic Beanstalk での環境のモニタリング

Elastic Beanstalk ヘルスモニタリングを使用すると、アプリケーションの可用性を検証し、メトリクスがしきい値を超えたときにアクティブ化するアラートを作成できます。コンソールとコマンドラインの両方で Elastic Beanstalk ヘルスモニタリングを使用して、環境のステータスを追跡できます。

トピック

- AWS マネジメントコンソールでの環境の状態のモニタリング
- EB CLI を使用した環境ヘルスのモニタリング
- ベーシックヘルスレポート
- Elastic Beanstalk でのヘルスレポートとモニタリングの強化
- アラームの管理
- Elastic Beanstalk 環境の変更履歴の表示
- Elastic Beanstalk 環境のイベントストリームの表示
- サーバーインスタンスの一覧表示と接続
- Elastic Beanstalk 環境の Amazon EC2 インスタンスからのログの表示

AWS マネジメントコンソールでの環境の状態のモニタリング

Elastic Beanstalk コンソールからアプリケーションの動作情報にアクセスできます。コンソールは、環境の状態とアプリケーションの状態を一目で分かるように表示します。コンソールの [環境] ページおよび各アプリケーションのページで、リスト上の環境はステータスを示すために色分けされています。

Elastic Beanstalk コンソールで環境を監視するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[モニタリング] を選択します。

[Monitoring] ページには、環境全体の統計（CPU 使用率や平均レイテンシー）が表示されます。全体の統計に加え、時間ごとのリソース使用率を示すモニタリンググラフも表示されます。グラフの任意の場所をクリックして、詳細情報を表示できます。

> **Note**
>
> デフォルトでは、基本的な CloudWatch メトリクスだけが有効化されていて、5 分周期でデータを返します。環境の設定を変更することで、より粒度の高い、1 分単位の CloudWatch メトリクスを有効化することもできます。

モニタリンググラフ

[モニタリング] ページには、環境のヘルスに関連するメトリクスの概要が表示されます。これには、Elastic Load Balancing および Amazon EC2 によって提供されるデフォルトのメトリクスセットと、環境の経時変化のグラフが含まれます。

グラフの上のバーには、選択できるさまざまな時間間隔が表示されます。例えば、先週の情報を表示するには、[1w] を選択します。または、[3h] を選択すると、過去 3 時間の情報が表示されます。

より幅広く時間間隔を選択するには、[カスタム] を選択します。ここからは、[絶対] または [相対] という 2 つの範囲オプションがあります。[絶対] オプションを使用すると、2023 年 1 月 1 日から 2023 年 6 月 30 日など、特定の日付範囲を指定できます。[相対] オプションを使用すると、特定の時間単位 ([分]、[時間]、[日]、[週]、または [月]) で整数を選択できます。[10 時間]、[10 日]、[10 か月] などです。

モニタリングコンソールのカスタマイズ

カスタムメトリクスを作成および表示するには、Amazon CloudWatch を使用する必要があります。CloudWatch を使用すると、単一のビューでリソースをモニタリングするために、カスタムダッシュボードを作成できます。[ダッシュボードに追加] を選択して、[モニタリング] ページから Amazon CloudWatch コンソールに移動します。Amazon CloudWatch では、新しいダッシュボードを作成するか、既存のダッシュボードを選択できます。詳細については、「Amazon CloudWatch ユーザーガイド」の「Amazon CloudWatch ダッシュボードの使用」を参照してください。

Elastic Load Balancing と Amazon EC2 メトリクスは、すべての環境で有効になります。

強化ヘルスを使用すると、EnvironmentHealth メトリクスが有効になり、グラフが 1 つモニタリングコンソールに自動的に追加されます。拡張ヘルスでは、マネジメントコンソールに [Health] ページも追加されます。強化ヘルスメトリクスのリストについては、「環境の Amazon CloudWatch カスタムメトリクスの発行」を参照してください。

EB CLI を使用した環境ヘルスのモニタリング

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) は、 AWS Elastic Beanstalk 環境を管理するためのコマンドラインツールです。EB CLI では、環境の状態をリアルタイムで、現在の Elastic Beanstalk コンソールより細かくモニタリングすることもできます。

EB CLI をインストールして設定したら、 eb create コマンドを使用して新しい環境を起動し、その環境にコードをデプロイできます。Elastic Beanstalk コンソールで既に環境を作成している場合は、プロジェクトフォルダ (空でも問題ありません) で eb init を実行し、プロンプトに従うことによって、その環境に EB CLI をアタッチできます。

> **Important**
>
> `pip install` オプションを付けて `--upgrade` を実行し、EB CLI の最新バージョンを使用していることを確認します。
>
> ```
> $ sudo pip install --upgrade awsebcli
> ```
>
> EB CLI の詳細なインストール手順については、「セットアップスクリプトを使用して EB CLI をインストールする (推奨)」を参照してください。

EB CLI を使用して対象環境のヘルスステータスを監視するには、eb init を実行して画面の指示に従うことで、まず、ローカルのプロジェクトフォルダを設定する必要があります。詳細な手順については、「EB CLI の設定」を参照してください。

Elastic Beanstalk で既に環境が実行されており、EB CLI を使用してその状態をモニタリングする場合は、既存の環境を使用するために、以下のステップに従って、それに EB CLI をアタッチします。

EB CLI を既存の環境にアタッチするには

1. コマンドラインターミナルを開き、ユーザーフォルダに移動します。
2. 対象環境用に新しいフォルダを作成して開きます。
3. eb init コマンドを実行し、モニタリングするアプリケーションと環境を選択します。選択したアプリケーションを実行している環境が 1 つしかない場合、EB CLI は自動的にそれを選択するので、次の例に示すように、手動で環境を選択する必要はありません。

```
~/project$ eb init
Select an application to use
```

```
1) elastic-beanstalk-example
2) [ Create new Application ]
(default is 2): 1
Select the default environment.
You can change this later by typing "eb use [environment_name]".
1) elasticBeanstalkEx2-env
2) elasticBeanstalkExa-env
(default is 1): 1
```

EB CLI を使用して状態をモニタリングするには

1. コマンドラインを開き、ユーザーフォルダに移動します。
2. eb health コマンドを実行して、お客様の環境内のインスタンスのヘルスステータスを表示します。この例では、5 つのインスタンスが Linux 環境内で実行されています。

```
~/project $ eb health
 elasticBeanstalkExa-env                                  Ok
 2015-07-08 23:13:20
WebServer
    Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0

  instance-id   status     cause
                                            health
    Overall     Ok
  i-d581497d    Ok
  i-d481497c    Ok
  i-136e00c0    Ok
  i-126e00c1    Ok
  i-8b2cf575    Ok

  instance-id   r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75
 p50     p10                                    requests
    Overall     671.8   100.0    0.0    0.0    0.0    0.003    0.002    0.001
 0.001   0.000
  i-d581497d    143.0    1430      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-d481497c    128.8    1288      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-136e00c0    125.4    1254      0      0      0    0.004    0.002    0.001
 0.001   0.000
```

```
  i-126e00c1    133.4    1334      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-8b2cf575    141.2    1412      0      0      0    0.003    0.002    0.001
 0.001   0.000

  instance-id   type       az   running     load 1  load 5      user%  nice%
 system%  idle%   iowait%                            cpu
  i-d581497d    t2.micro   1a   12 mins        0.0    0.04        6.2    0.0
 1.0   92.5       0.1
  i-d481497c    t2.micro   1a   12 mins       0.01    0.09        5.9    0.0
 1.6   92.4       0.1
  i-136e00c0    t2.micro   1b   12 mins       0.15    0.07        5.5    0.0
 0.9   93.2       0.0
  i-126e00c1    t2.micro   1b   12 mins       0.17    0.14        5.7    0.0
 1.4   92.7       0.1
  i-8b2cf575    t2.micro   1c   1 hour        0.19    0.08        6.5    0.0
 1.2   92.1       0.1

  instance-id   status     id   version              ago
                                        deployments
  i-d581497d    Deployed   1    Sample Application   12 mins
  i-d481497c    Deployed   1    Sample Application   12 mins
  i-136e00c0    Deployed   1    Sample Application   12 mins
  i-126e00c1    Deployed   1    Sample Application   12 mins
  i-8b2cf575    Deployed   1    Sample Application   1 hour
```

この例では、1 つのインスタンスが Windows 環境内で実行されています。

```
~/project $ eb health
 WindowsSampleApp-env                                  Ok
      2018-05-22 17:33:19
WebServer                                                 IIS 10.0 running on 64bit
 Windows Server 2016/2.2.0
  total      ok    warning  degraded  severe    info   pending  unknown
    1        1        0        0        0        0        0        0

  instance-id           status     cause
                                              health
    Overall             Ok
  i-065716fba0e08a351   Ok

  instance-id           r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90
 p75     p50     p10                          requests
```

```
    Overall              13.7   100.0    0.0    0.0    0.0    1.403    0.970
0.710   0.413   0.079
 i-065716fba0e08a351     2.4   100.0    0.0    0.0    0.0    1.102*   0.865
0.601   0.413   0.091

 instance-id           type       az   running     % user time    % privileged
time  % idle time                                  cpu
 i-065716fba0e08a351   t2.large   1b   4 hours             0.2
0.1         99.7

 instance-id           status     id   version              ago
                                       deployments
 i-065716fba0e08a351   Deployed   2    Sample Application   4 hours
```

出力の読み取り

出力は、環境の名前、環境の総合的な状態、および現在の日付を画面の上部に表示します。

```
elasticBeanstalkExa-env                                  Ok
 2015-07-08 23:13:20
```

その次の 3 行には、環境のタイプ (ここでは "WebServer")、設定 (Ruby 2.1 with Puma)、および 7 つの状態にあるインスタンスの数が表示されます。

```
WebServer
 Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0
```

出力の残りは 4 つのセクションに分かれています。最初のセクションには、環境全体と各インスタンスのステータスとそのステータスの原因が表示されます。以下の例は、環境内のステータスが Info である 2 つのインスタンスと、デプロイが開始されたことを示す原因を示しています。

```
  instance-id    status     cause
                                       health
    Overall      Ok
  i-d581497d     Info       Performing application deployment (running for 3 seconds)
  i-d481497c     Info       Performing application deployment (running for 3 seconds)
  i-136e00c0     Ok
  i-126e00c1     Ok
```

```
  i-8b2cf575     Ok
```

ヘルスのステータスと色の詳細については、「状態の色とステータス」を参照してください。

[requests] セクションには、各インスタンスに関するウェブサーバーログからの情報が表示されます。この例では、各インスタンスは正常にリクエストを受け取っており、エラーはありません。

```
  instance-id    r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75     p50
    p10                                requests
    Overall      13.7    100.0    0.0    0.0    0.0    1.403    0.970    0.710   0.413
  0.079
  i-d581497d     2.4     100.0    0.0    0.0    0.0    1.102*   0.865    0.601   0.413
  0.091
  i-d481497c     2.7     100.0    0.0    0.0    0.0    0.842*   0.788    0.480   0.305
  0.062
  i-136e00c0     4.1     100.0    0.0    0.0    0.0    1.520*   1.088    0.883   0.524
  0.104
  i-126e00c1     2.2     100.0    0.0    0.0    0.0    1.334*   0.791    0.760   0.344
  0.197
  i-8b2cf575     2.3     100.0    0.0    0.0    0.0    1.162*   0.867    0.698   0.477
  0.076
```

[CPU] セクションには、各インスタンスのオペレーティングシステムメトリクスが表示されます。出力はオペレーティングシステムによって異なります。Linux 環境での出力を次に示します。

```
  instance-id   type       az   running     load 1  load 5      user%  nice%  system%
 idle%   iowait%                              cpu
  i-d581497d    t2.micro   1a   12 mins        0.0    0.03        0.2    0.0      0.0
 99.7       0.1
  i-d481497c    t2.micro   1a   12 mins        0.0    0.03        0.3    0.0      0.0
 99.7       0.0
  i-136e00c0    t2.micro   1b   12 mins        0.0    0.04        0.1    0.0      0.0
 99.9       0.0
  i-126e00c1    t2.micro   1b   12 mins       0.01    0.04        0.2    0.0      0.0
 99.7       0.1
  i-8b2cf575    t2.micro   1c   1 hour         0.0    0.01        0.2    0.0      0.1
 99.6       0.1
```

Windows 環境での出力を次に示します。

```
  instance-id           type       az   running     % user time    % privileged time  %
 idle time
```

```
  i-065716fba0e08a351   t2.large   1b   4 hours            0.2                 0.0
      99.8
```

サーバーとオペレーティングシステムのメトリクスの詳細については、「インスタンスメトリクス」を参照してください。

最終セクション [deployments] には、各インスタンスのデプロイステータスが表示されます。ローリングデプロイが失敗した場合は、表示されたデプロイ ID、ステータス、バージョンラベルを使って、間違ったバージョンを実行している環境のインスタンスを特定できます。

```
  instance-id   status     id   version              ago
                                   deployments
  i-d581497d    Deployed   1    Sample Application   12 mins
  i-d481497c    Deployed   1    Sample Application   12 mins
  i-136e00c0    Deployed   1    Sample Application   12 mins
  i-126e00c1    Deployed   1    Sample Application   12 mins
  i-8b2cf575    Deployed   1    Sample Application   1 hour
```

インタラクティブヘルスビュー

eb health コマンドは、環境の状態のスナップショットを表示します。表示される情報が 10 秒ごとに更新されるようにするには、`--refresh` オプションを使用します。

```
$ eb health --refresh
 elasticBeanstalkExa-env                            Ok
 2015-07-09 22:10:04 (1 secs)
WebServer
          Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0

  instance-id   status     cause
                                        health
    Overall     Ok
  i-bb65c145    Ok         Application deployment completed 35 seconds ago and took 26
 seconds
  i-ba65c144    Ok         Application deployment completed 17 seconds ago and took 25
 seconds
  i-f6a2d525    Ok         Application deployment completed 53 seconds ago and took 26
 seconds
  i-e8a2d53b    Ok         Application deployment completed 32 seconds ago and took 31
 seconds
```

```
 i-e81cca40    Ok

 instance-id   r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75     p50
  p10                                  requests
   Overall     671.8   100.0    0.0    0.0    0.0    0.003    0.002    0.001   0.001
0.000
 i-bb65c145    143.0    1430      0      0      0    0.003    0.002    0.001   0.001
0.000
 i-ba65c144    128.8    1288      0      0      0    0.003    0.002    0.001   0.001
0.000
 i-f6a2d525    125.4    1254      0      0      0    0.004    0.002    0.001   0.001
0.000
 i-e8a2d53b    133.4    1334      0      0      0    0.003    0.002    0.001   0.001
0.000
 i-e81cca40    141.2    1412      0      0      0    0.003    0.002    0.001   0.001
0.000

 instance-id   type       az   running     load 1  load 5      user%  nice%  system%
idle%   iowait%                              cpu
 i-bb65c145    t2.micro   1a   12 mins        0.0    0.03        0.2    0.0      0.0
99.7       0.1
 i-ba65c144    t2.micro   1a   12 mins        0.0    0.03        0.3    0.0      0.0
99.7       0.0
 i-f6a2d525    t2.micro   1b   12 mins        0.0    0.04        0.1    0.0      0.0
99.9       0.0
 i-e8a2d53b    t2.micro   1b   12 mins       0.01    0.04        0.2    0.0      0.0
99.7       0.1
 i-e81cca40    t2.micro   1c   1 hour         0.0    0.01        0.2    0.0      0.1
99.6       0.1

 instance-id   status     id   version              ago
                                 deployments
 i-bb65c145    Deployed   1    Sample Application   12 mins
 i-ba65c144    Deployed   1    Sample Application   12 mins
 i-f6a2d525    Deployed   1    Sample Application   12 mins
 i-e8a2d53b    Deployed   1    Sample Application   12 mins
 i-e81cca40    Deployed   1    Sample Application   1 hour

(Commands: Help,Quit, # # # #)
```

この例は、最近、インスタンスを 1 個から 5 個にスケールアップした環境を示しています。スケーリングオペレーションが完了し、すべてのインスタンスがヘルスチェックに合格したら、リクエスト

を受け取ることができるようになります。対話モードでは、ヘルスステータスが 10 秒ごとに更新されます。右上隅で、タイマーが次の更新までカウントダウンされます。

左下隅で、レポートがオプションのリストを表示します。インタラクティブモードを終了するには、Q キーを押します。スクロールするには、矢印キーを押します。他のコマンドのリストを表示するには、H キーを押します。

インタラクティブヘルスビューのオプション

環境の状態をインタラクティブに表示する場合、キーボードのキーを使用して、表示を調整したり、個々のインスタンスを置換または再起動するように Elastic Beanstalk に指示したりできます。インタラクティブモードでヘルスレポートを表示しているときに使用できるコマンドのリストを表示するには、H キーを押します。

```
  up,down,home,end   Scroll vertically
  left,right         Scroll horizontally
  F                  Freeze/unfreeze data
  X                  Replace instance
  B                  Reboot instance
  <,>                Move sort column left/right
  -,+                Sort order descending/ascending
  P                  Save health snapshot data file
  Z                  Toggle color/mono mode
  Q                  Quit this program

  Views
  1                  All tables/split view
  2                  Status Table
  3                  Request Summary Table
  4                  CPU%/Load Table
  H                  This help menu


(press Q or ESC to return)
```

ベーシックヘルスレポート

このトピックでは、Elastic Beanstalk の基本ヘルスが提供する機能について説明します。

AWS Elastic Beanstalk は、複数のソースからの情報を使用して、環境が利用可能かどうかを判断し、インターネットからのリクエストを処理します。環境の状態は 4 つの色のい

ずれかで表され、Elastic Beanstalk コンソールの [環境の概要] ページに表示されます。また、DescribeEnvironments API から、および EB CLI を使用して eb status を呼び出すことによっても利用できます。

基本的なヘルスレポートシステムは、Elastic Beanstalk 環境におけるインスタンスのヘルスに関する情報を提供します。ロードバランシング環境の場合は Elastic Load Balancing、単一インスタンス環境の場合は Amazon Elastic Compute Cloud によって実行されるヘルスチェックに基づいて、Elastic Beanstalk 環境におけるインスタンスの状態に関する情報を提供します。

EC2 インスタンスのヘルスチェックに加えて、Elastic Beanstalk は、対象環境内のその他のリソースのモニタリングも行い、不足しているリソースや、誤設定のためユーザーに使用不可となったリソースをレポートします。

対象環境内のリソースによって収集されたメトリクスは、5 分間隔で Amazon CloudWatch に発行されます。これには、EC2 からのオペレーティングシステムメトリクス、Elastic Load Balancing からのリクエストメトリクスが含まれます。環境コンソールの [Monitoring (モニタリング)] ページで、これらの CloudWatch メトリクスに基づいてグラフを表示できます。基本ヘルスの場合、これらのメトリクスは環境のヘルスステータスを判断するために使用されません。

トピック

- ヘルスステータスの色
- Elastic Load Balancing のヘルスチェック
- 単一インスタンスおよびワーカー枠環境のヘルスチェック
- 追加のチェック
- Amazon CloudWatch メトリクス

ヘルスステータスの色

Elastic Beanstalk は、ウェブサーバーの環境のヘルスステータスを、そのサーバーで実行中のアプリケーションによるヘルスチェックへの応答に基づいてレポートします。Elastic Beanstalk は、以下の表に示しているように、4 色のいずれかでステータスを表します。

カラー	説明
Grey	環境が更新中です。

カラー	説明
Green	環境が最新のヘルスチェックで合格になりました。環境内の少なくとも 1 つのインスタンスが使用可能であり、リクエストを受け取っています。
黄色	対象環境が 1 つ以上のヘルスチェックで失格になりました。環境へのいくつかのリクエストが失敗しています。
赤	対象環境が 3 つ以上のヘルスチェックで失格になったか、環境のリソースが使用不可になっています。リクエストは一貫して失敗しています。

これらの説明は、基本ヘルスレポートを使用している環境にのみ適用されます。拡張ヘルスの詳細については、「状態の色とステータス」を参照してください。

Elastic Load Balancing のヘルスチェック

負荷が分散されている環境では、インスタンスが正常であることを確認するために、Elastic Load Balancing は環境内の各インスタンスに 10 秒ごとにリクエストを送信します。デフォルトでは、ロードバランサーはポート 80 で TCP 接続を開くように設定されています。インスタンスが接続に応答した場合、そのインスタンスは正常と見なされます。

アプリケーションで既存のリソースを指定することによって、この設定をオーバーライドすることもできます。`/health` などのパスを指定した場合、ヘルスチェック URL は `HTTP:80/health` に設定されます。ヘルスチェック URL は、常にアプリケーションによって処理されるパスに設定する必要があります。アプリケーションの前にあるウェブサーバーによって処理またはキャッシュされる静的なページに設定すると、ヘルスチェックはアプリケーションサーバーまたはウェブコンテナで発生する問題を検出しません。ヘルスチェック URL の変更方法については、ヘルスチェック を参照してください。

ヘルスチェック URL が設定されている場合、Elastic Load Balancing では、送信する GET リクエストで `200 OK` の応答が返される必要があります。5 秒以内に応答が返されなかった場合、または他の HTTP ステータスコードで応答が返された場合、アプリケーションはヘルスチェックで失格になります。5 回連続してヘルスチェックで失格になった後、Elastic Load Balancing はそのインスタンスをサービスから除外します。

Elastic Load Balancing ヘルスチェックの詳細については、Elastic Load Balancing ユーザーガイドの「ヘルスチェック」を参照してください。

> **Note**
>
> ヘルスチェック URL を設定しても、環境の Auto Scaling グループのヘルスチェック動作は変わりません。正常でないインスタンスはロードバランサーから削除されますが、インスタンス置き換えの判断基準として Elastic Load Balancing ヘルスチェックを使用するように Amazon EC2 Auto Scaling を設定していない場合、そのインスタンスは Amazon EC2 Auto Scaling によって自動的に置き換えられることはありません。Elastic Load Balancing ヘルスチェックに失敗したインスタンスを置き換えるように Amazon EC2 Auto Scaling を設定するには、「Elastic Beanstalk 環境の Auto Scaling ヘルスチェック設定」を参照してください。

単一インスタンスおよびワーカー枠環境のヘルスチェック

単一インスタンスまたはワーカー枠環境では、Elastic Beanstalk は Amazon EC2 インスタンスのステータスをモニタリングして、インスタンスの状態を決定します。HTTP ヘルスチェック URL を含む Elastic Load Balancing のヘルス設定は、これらの環境タイプでは使用できません。

Amazon EC2 インスタンスのステータスチェックの詳細については、「Amazon EC2 ユーザーガイド」の「ステータスチェックでインスタンスをモニタリングする」を参照してください。

追加のチェック

Elastic Load Balancing ヘルスチェックに加えて、Elastic Beanstalk は、対象環境内のリソースのモニタリングを行い、デプロイに失敗したリソースや、誤設定されたリソース、使用不可になったリソースが見つかった場合、そのリソースのヘルスステータスを赤色に変更します。これらのチェックでは、以下のことが確認されます。

- 環境の Auto Scaling グループと少なくとも 1 つのインスタンスが使用可能である。
- 環境のセキュリティグループが使用可能で、ポート 80 で受信トラフィックを許可するように設定されている。
- 環境 CNAME が存在し、正しいロードバランサーを指している。
- ワーカー環境では、Amazon Simple Queue Service (Amazon SQS) キューは少なくとも 3 分に 1 回ポーリングされています。

Amazon CloudWatch メトリクス

基本的なヘルスレポートでは、Elastic Beanstalk サービスは Amazon CloudWatch にメトリクスを公開しません。環境コンソールの [Monitoring (モニタリング)] ページでグラフの生成に使用される CloudWatch メトリクスは、対象環境内のリソースによって発行されます。

たとえば、EC2 は対象環境の Auto Scaling グループのインスタンスについて以下のメトリクスを発行します。

`CPUUtilization`

現在使用中のコンピューティングユニットのパーセンテージ。

`DiskReadBytes`, `DiskReadOps`, `DiskWriteBytes`, `DiskWriteOps`

読み取りおよび書き込みバイトの数、読み取りおよび書き込みオペレーションの数。

`NetworkIn`, `NetworkOut`

送信および受信バイトの数。

Elastic Load Balancing は対象環境のロードバランサーについて以下のメトリクスを発行します。

`BackendConnectionErrors`

対象環境のロードバランサーとインスタンスとの接続失敗の数。

`HTTPCode_Backend_2XX`, `HTTPCode_Backend_4XX`

対象環境内のインスタンスによって生成された成功 (2XX) とクライアントエラー (4XX) の応答コードの数。

`Latency`

ロードバランサーがインスタンスにリクエストを中継してから応答を受信するまでの秒数。

`RequestCount`

完了したリクエストの数。

これらのリストにはすべてのメトリクスが含まれているわけではありません。これらのリソースについてレポート可能なメトリクスの完全なリストについては、Amazon CloudWatch 開発者ガイドの以下のトピックを参照してください。

メトリクス

名前空間	トピック
AWS::ElasticLoadBalancing::LoadBalancer	Elastic Load Balancing のメトリクスとリソース
AWS::AutoScaling::AutoScalingGroup	Amazon Elastic Compute Cloud のメトリクスとリソース
AWS::SQS::Queue	Amazon SQS メトリクスおよびリソース
AWS::RDS::DBInstance	Amazon RDS のディメンションおよびメトリクス

ワーカー環境ヘルスメトリクス

ワーカー環境でのみ、SQS デーモンは CloudWatch に環境ヘルスのカスタムメトリクスを発行します (値 1 は Green です)。`ElasticBeanstalk/SQSD` 名前空間を使用してアカウントの CloudWatch ヘルスメトリクスデータを確認できます。メトリクスディメンションは `EnvironmentName`、メトリクス名は `Health` です。すべてのインスタンスが、同じ名前空間にメトリクスを公開します。

デーモンがメトリクスを発行できるように、環境のインスタンスプロファイルに、`cloudwatch:PutMetricData` を呼び出す権限を付与する必要があります。この権限は、デフォルトのインスタンスプロファイルに含まれます。詳細については、「」を参照してください[Elastic Beanstalk インスタンスプロファイルの管理](#)

Elastic Beanstalk でのヘルスレポートとモニタリングの強化

このセクションでは、Elastic Beanstalk 拡張ヘルス機能について説明します。

拡張ヘルスレポートは、環境で を有効にして、 が環境内のリソースに関する追加情報 AWS Elastic Beanstalk を収集できるようにする機能です。Elastic Beanstalk は、収集された情報を分析して環境全体の状態をより的確に示し、アプリケーションの使用を妨げる可能性のある問題を特定するために役立ちます。

色で状態を示す機能が変更されたことに加え、拡張ヘルスレポートにはステータス記述子が追加されています。これは、環境の状態が黄色または赤色の場合に検出された問題の重大度を示します。現在

のステータスに関する詳細情報があるときは、[Causes] ボタンを選択して、[Health] ページにヘルスに関する詳細情報を表示できます。

環境内で実行されている Amazon EC2 インスタンスに関する詳細なヘルス情報を提供するため、Elastic Beanstalk は、拡張ヘルスをサポートする各プラットフォームのバージョンの Amazon マシンイメージ (AMI) にヘルスエージェントを含めます。状態エージェントは、ウェブサーバーログとシステムメトリクスを監視して、Elastic Beanstalk サービスに中継します。Elastic Beanstalk は、これらのメトリクスと、Elastic Load Balancing と Amazon EC2 Auto Scaling から取られたデータを分析し、環境の状態に関する全体像を提示します。

環境のリソースに関する情報の収集および提示に加えて、Elastic Beanstalk は何種類かのエラー状態に備えて環境内のリソースを監視し、通知を提供します。これは、障害を回避し、設定の問題を解決するために役立ちます。お客様の環境のヘルスに影響を与える要因としては、アプリケーションによって処理された各リクエストの結果、お客様のインスタンスのオペレーティングシステムからのメトリクス、最新のデプロイのステータスなどがあります。

Elastic Beanstalk コンソールで環境の概要ページを使用するか、Elastic Beanstalk コマンドラインインターフェイス (EB CLI) で eb health コマンドを実行することで、ヘルスステータスをリアルタイムで確認できます。Elastic Beanstalk によって収集された拡張ヘルスレポートのための情報を Amazon CloudWatch にカスタムメトリクスとしてパブリッシュするように環境を設定することで、環境とインスタンスの状態を記録し、追跡することができます。無料の `EnvironmentHealth` 以外のすべてのメトリクスには、カスタムメトリクスに対する CloudWatch 料金がかかります。

> **Windows プラットフォームのメモ**
>
> Windows Server 環境で拡張ヘルスレポートを有効にするときは、IIS ログ設定は変更しないでください。拡張ヘルスモニタリングを正しく動作させるには、IIS のログ記録を W3C 形式および ETW イベントのみまたはログファイルと ETW イベントの両方のログイベント送信先で設定する必要があります。
> さらに、いずれの環境のインスタンスでも、Elastic Beanstalk ヘルスエージェントの Windows サービスを無効化または停止しないでください。インスタンスで拡張ヘルス情報を収集および報告するには、このサービスが有効で実行中である必要があります。

初めて環境を作成すると、必要なロールを作成するよう Elastic Beanstalk から求められ、デフォルトで拡張ヘルスレポートが有効になります。拡張ヘルスレポートの詳しいしくみについては、この後の説明を参照してください。すぐに開始するには、「Elastic Beanstalk の拡張ヘルスレポートの有効化」を参照してください。

トピック

- Elastic Beanstalk のヘルスエージェント
- インスタンスと環境の状態を判断するための要素
- ヘルスチェックルールのカスタマイズ
- 拡張ヘルスレポートのロール
- 拡張ヘルス認可
- 拡張ヘルスレポートのイベント
- 更新、デプロイ、およびスケーリング中の拡張ヘルスレポートの動作
- Elastic Beanstalk の拡張ヘルスレポートの有効化
- 環境管理コンソールでの拡張ヘルスモニタリング
- 状態の色とステータス
- インスタンスメトリクス
- 環境の拡張ヘルスルールの設定
- 環境の Amazon CloudWatch カスタムメトリクスの発行
- Elastic Beanstalk API での拡張ヘルスレポートの使用
- 拡張ヘルスログ形式
- 通知とトラブルシューティング

Elastic Beanstalk のヘルスエージェント

Elastic Beanstalk のヘルスエージェントは、環境内の各 Amazon EC2 インスタンスで実行されるデーモンプロセス (または Windows 環境のサービス) であり、オペレーティングシステムおよびアプリケーションレベルのヘルスメトリクスをモニタリングし、問題を Elastic Beanstalk に報告します。ヘルスエージェントは、各プラットフォームのバージョン 2.0 以降、すべてのプラットフォームのバージョンに含まれています。

状態エージェントからの報告内容は、ベーシックヘルスレポートの一部として Amazon EC2 Auto Scaling および Elastic Load Balancing によって CloudWatch に公開されるメトリクスと似ており、CPU 負荷、HTTP コード、レイテンシーなどが含まれます。ただし、状態エージェントによるレポートは、ベーシックヘルスレポートより高い詳細度と頻度で直接 Elastic Beanstalk に送信されます。

ベーシックヘルスレポートの場合、これらのメトリクスは、5 分間隔で公開され、環境マネジメントコンソールのグラフで確認できます。拡張ヘルスでは、Elastic Beanstalk ヘルスエージェントは 10

秒ごとに Elastic Beanstalk にメトリクスを報告します。Elastic Beanstalk は状態エージェントから提供されたメトリクスを使用して、環境内の各インスタンスのヘルスステータスを判断します。さらに、他の要素と組み合わせて、環境全体の状態を判断します。

環境全体の状態は、Elastic Beanstalk によって 60 秒間隔で CloudWatch に公開され、Elastic Beanstalk コンソールの環境の概要ページでリアルタイムで確認できます。状態エージェントによって報告された詳細なメトリクスは、EB CLI の eb health コマンドを使用してリアルタイムで確認できます。

個々のインスタンスおよび環境レベルのメトリクスを 60 秒間隔で CloudWatch に公開することもできます (追加料金が必要です)。CloudWatch に公開されたメトリクスを使用すると、環境マネジメントコンソールでモニタリンググラフを作成できます。

拡張ヘルスレポートで料金が発生するのは、拡張ヘルスレポートのメトリクスを CloudWatch に公開するオプションを選択した場合のみです。拡張ヘルスレポートを使用していて、拡張ヘルスレポートのメトリクスを公開するオプションを選択していなくても、ベーシックヘルスレポートのメトリクスは無料で公開できます。

状態エージェントによって報告されるメトリクスの詳細については、「インスタンスメトリクス」を参照してください。拡張ヘルスメトリクスのメトリクスを CloudWatch に公開する方法の詳細については、「」を参照してください環境の Amazon CloudWatch カスタムメトリクスの発行

インスタンスと環境の状態を判断するための要素

Elastic Beanstalk 拡張ヘルスレポートは、基本的なヘルスレポートのシステムチェック (Elastic Load Balancing のヘルスチェック および リソースモニタリングなど) に加えて、環境内のインスタンスの状態に関する追加のデータを収集します。これには、オペレーティングシステムのメトリクス、サーバーログ、およびデプロイや更新などの進行中の環境オペレーションの状態が含まれます。Elastic Beanstalk ヘルスレポートサービスは、利用可能なすべてのソースからの情報を組み合わせて分析し、環境全体の状態を判断します。

オペレーションとコマンド

環境内でオペレーション (アプリケーションの新しいバージョンのデプロイなど) を実行する場合、Elastic Beanstalk では、環境のヘルスステータスに影響するいくつかの変更が行われます。

たとえば、複数のインスタンスを実行する環境にアプリケーションの新しいバージョンをデプロイする場合、EB CLI で環境の状態をモニタリングしていると、次のようなメッセージが表示されることがあります。

```
  id              status     cause
    Overall       Info       Command is executing on 3 out of 5 instances
  i-bb65c145      Pending    91 % of CPU is in use. 24 % in I/O wait
                             Performing application deployment (running for 31 seconds)
  i-ba65c144      Pending    Performing initialization (running for 12 seconds)
  i-f6a2d525      Ok         Application deployment completed 23 seconds ago and took 26
 seconds
  i-e8a2d53b      Pending    94 % of CPU is in use. 52 % in I/O wait
                             Performing application deployment (running for 33 seconds)
  i-e81cca40      Ok
```

この例では、環境全体のステータスが Ok であり、このステータスの原因は Command is executing on 3 out of 5 instances です。環境内の 3 つのインスタンスのステータスが Pending であり、オペレーションが進行中であることを示します。

オペレーションが完了すると、Elastic Beanstalk により、オペレーションに関する追加情報が報告されます。たとえば、アプリケーションの新しいバージョンに更新が完了しているインスタンスについては、次の情報が表示されます。

```
i-f6a2d525      Ok         Application deployment completed 23 seconds ago and took 26
 seconds
```

インスタンスのヘルスに関する情報には、お客様の環境内の各インスタンスへの最新のデプロイに関する詳細も含まれます。各インスタンスについて、デプロイ ID とステータスがレポートされます。デプロイ ID は整数であり、アプリケーションの新しいバージョンをデプロイするか、環境変数などインスタンス関連の設定オプションの設定を変更するたびに、1 ずつ増えます。デプロイに関する情報を使用して、ローリングデプロイの失敗後にアプリケーションの間違ったバージョンを実行しているインスタンスを特定できます。

Elastic Beanstalk は、原因を示す列に、成功したオペレーションに関する情報メッセージと、複数のヘルスチェックにおけるその他のヘルスステータスを表示しますが、これらは無期限に保存されるわけではありません。環境の異常な状態の原因を示す情報が保存されるのは、環境の状態が正常に戻るまでです。

コマンドタイムアウト

インスタンスが正常な状態に移行できるように、Elastic Beanstalk では、オペレーションが開始した時点からコマンドタイムアウトが適用されます。このコマンドタイムアウトは、環境の更新およびデプロイ設定 (aws:elasticbeanstalk:command 名前空間) で設定されており、デフォルト値は 10 分です。

ローリング更新の実行中、Elastic Beanstalk は、オペレーション内の各バッチに別々のタイムアウトを適用します。このタイムアウトは環境のローリング更新の一部として (aws:autoscaling:updatepolicy:rollingupdate 名前空間で) 設定されます。バッチ内のすべてのインスタンスがローリング更新タイムアウト内で正常に実行されている場合は、オペレーションが次のバッチに進みます。それ以外の場合、オペレーションは失敗となります。

> **Note**
>
> アプリケーションがヘルスチェックで OK ステータスにならなくても、別のレベルで安定する場合は、`HealthCheckSuccessThreshold` で `aws:elasticbeanstalk:command` namespace オプションを設定して、Elastic Beanstalk でインスタンスが正常と見なされるレベルに変更できます。

ウェブサーバー環境が正常であると見なされるには、環境内またはバッチ内の各インスタンスが 2 分間にわたって連続して行われる 12 のヘルスチェックに合格する必要があります。ワーカー枠環境の場合は、各インスタンスが 18 のヘルスチェックに合格する必要があります。Elastic Beanstalk は、ヘルスチェックで異常が検出されても、コマンドタイムアウトの前には、環境のヘルスステータスを引き下げません。環境内のインスタンスがコマンドタイムアウト内に正常な状態である場合、オペレーションは成功となります。

HTTP リクエスト

環境で進行中のオペレーションがない場合、インスタンスと環境の状態に関する情報のプライマリソースは、各インスタンスのウェブサーバーログです。インスタンスの状態と環境全体の状態を判断するためには、リクエストの数、各リクエストの結果、各リクエストが解決された速度が考慮されます。

Linux ベースのプラットフォームでは、Elastic Beanstalk はウェブサーバーのログを読み取り、解析して HTTP リクエストに関する情報を取得します。Windows Server プラットフォームでは、Elastic Beanstalk はこの情報を IIS ウェブサーバーから直接受け取ります。

環境にはアクティブなウェブサーバーがない可能性があります。たとえば、複数コンテナ Docker プラットフォームにはウェブサーバーは含まれません。その他のプラットフォームにはウェブサーバーがあり、アプリケーションがこれを無効にする可能性があります。このような場合、環境では、ヘルス情報を Elastic Beanstalk サービスに中継するために必要な形式のログを Elastic Beanstalk ヘルスエージェントに提供するための、追加の設定が必要です。詳細については、「拡張ヘルスログ形式」を参照してください。

オペレーティングシステムのメトリクス

Elastic Beanstalk は、状態エージェントから報告されたオペレーティング システムのメトリクスを監視し、継続的にシステムリソースが不足しているインスタンスを特定します。

状態エージェントによって報告されるメトリクスの詳細については、「インスタンスメトリクス」を参照してください。

ヘルスチェックルールのカスタマイズ

Elastic Beanstalk 拡張ヘルスレポートは、環境のヘルスを判断するための一連のルールに依存しています。これらのルールの一部は、特定のアプリケーションに適していない場合があります。よくあるケースは、仕様により頻繁に HTTP 4xx エラーを返すアプリケーションです。Elastic Beanstalk は、デフォルトのルールの 1 つを使用して、何かが間違っていると判断すると、エラーレートに応じて、環境のヘルスステータスを、OK から警告、パフォーマンス低下、または重大へと変化させます。このケースを正しく処理するために、Elastic Beanstalk ではこのルールを正しく設定して、アプリケーション HTTP 4xx エラーを無視するように設定できます。詳細については、「環境の拡張ヘルスルールの設定」を参照してください。

拡張ヘルスレポートのロール

拡張ヘルスレポートには、2 つのロールが必要です。Elastic Beanstalk 用のサービスロールと、環境用のインスタンスプロファイルです。サービスロールを使用すると、Elastic Beanstalk はユーザーに代わって他の AWS サービスとやり取りして、環境内のリソースに関する情報を収集できます。インスタンスプロファイルを使用すると、環境内のインスタンスがログを Amazon S3 に書き込んだり、拡張ヘルス情報を Elastic Beanstalk サービスに伝達したりできます。

Elastic Beanstalk コンソールまたは EB CLI を使用して Elastic Beanstalk 環境を作成すると、Elastic Beanstalk はデフォルトのサービスロールを作成し、必要な管理ポリシーを環境のデフォルトのインスタンスプロファイルにアタッチします。

API、 SDK、または を使用して環境を作成する場合は、これらのロール AWS CLI を事前に作成し、環境の作成時に指定して拡張ヘルスを使用する必要があります。環境に適切なロールを作成する方法については、「」を参照してくださいElastic Beanstalk サービスロール、インスタンスプロファイル、ユーザーポリシー

インスタンスプロファイルとサービスロールには管理ポリシーを使用することをお勧めします。管理ポリシーは、Elastic Beanstalk が管理する AWS Identity and Access Management (IAM) ポリシーです。管理ポリシーを使用すると、環境が適切に機能するために必要なすべてのアクセス許可を持つことが保証されます。

インスタンスプロファイルでは、ウェブサーバー層またはワーカー層環境に対して、AWSElasticBeanstalkWebTier または AWSElasticBeanstalkWorkerTier 管理ポリシーをそれぞれ使用できます。これら 2 つのマネージドインスタンスプロファイルポリシーの詳細については、「the section called “インスタンスプロファイル”」を参照してください。

拡張ヘルス認可

Elastic Beanstalk インスタンスプロファイルマネージドポリシーには、elasticbeanstalk:PutInstanceStatistics アクションに対するアクセス許可が含まれています。このアクションは Elastic Beanstalk API の一部ではありません。これは、環境インスタンスが拡張ヘルス情報を Elastic Beanstalk サービスに伝達するために内部的に使用する別の API の一部です。この API を直接呼び出すことはありません。

新しい環境を作成すると、elasticbeanstalk:PutInstanceStatistics アクションに対する認可がデフォルトで有効になっています。環境のセキュリティを強化し、ユーザーに代わってヘルスデータのスプーフィングを防ぐために、このアクションに対する認可を有効にしておくことをお勧めします。インスタンスプロファイルでマネージドポリシーを使用する場合、この機能は追加設定なしで新しい環境で使用できます。管理ポリシーの代わりにカスタムインスタンスプロファイルを使用すると、環境に [No Data] (データがありません) のヘルスステータスが表示されることがあります。これは、インスタンスで拡張ヘルスデータをサービスに伝達するアクションが許可されていないために発生します。

アクションを認可するには、インスタンスプロファイルに次のステートメントを含めます。

```
    {
      "Sid": "ElasticBeanstalkHealthAccess",
      "Action": [
        "elasticbeanstalk:PutInstanceStatistics"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:*:*:application/*",
        "arn:aws:elasticbeanstalk:*:*:environment/*"
      ]
    }
```

拡張ヘルス認可を現時点で使用しない場合は、the section called “aws:elasticbeanstalk:healthreporting:system” 名前空間の EnhancedHealthAuthEnabled オプションを false に設定して無効にします。このオプションが無効になっている場合、前述のアクセ

ス許可は必要ありません。アプリケーションと環境への最小特権アクセスのために、これらをインスタンスプロファイルから削除できます。

> **Note**
>
> 以前の `EnhancedHealthAuthEnabled` のデフォルト設定は `false` であったため、`elasticbeanstalk:PutInstanceStatistics` アクションに対する認可でもデフォルトでは無効となっています。既存の環境でこのアクションを有効にするには、the section called “aws:elasticbeanstalk:healthreporting:system” 名前空間の `EnhancedHealthAuthEnabled` オプションを `true` に設定します。このオプションは、設定ファイルのオプション設定を使用して設定できます。

拡張ヘルスレポートのイベント

拡張ヘルスレポートのシステムでは、環境の状態が変化するとイベントが生成されます。次の例は、環境の状態が Info、OK、Severe の間で変化した場合に出力されたイベントを示しています。

現在より悪い状態に変化した場合は、変化の原因を示すメッセージが拡張ヘルスイベントに含められます。

インスタンスレベルでのステータスの変化がすべて Elastic Beanstalk によるイベントの出力になるわけではありません。Elastic Beanstalk では、誤ったアラームを回避するために、複数のチェックで同じ問題が生じている場合のみ、状態に関連したイベントを出力します。

ステータス、色、原因など、環境レベルのリアルタイムの状態情報は、Elastic Beanstalk コンソールの環境の概要ページおよび EB CLI から利用できます。EB CLI を環境にアタッチして eb health コマンドを実行すると、環境内のインスタンスに関するリアルタイムのステータスを表示することもできます。

更新、デプロイ、およびスケーリング中の拡張ヘルスレポートの動作

拡張ヘルスレポートを有効にすると、設定更新中およびデプロイ中の環境の動作に影響が及ぶ可能性があります。Elastic Beanstalk は、すべてのインスタンスが一貫してヘルスチェックに合格するまで、更新のバッチを完了しません。また、拡張ヘルスレポートはより高い標準をヘルスに適用し、より多くの要素をモニタリングするため、基本ヘルスレポートの ELB ヘルスチェックに合格したインスタンスが、必ずしも拡張ヘルスレポートに合格するとは限りません。ヘルスチェックがアップデー

ト処理にどのように影響するかについては、ローリング設定更新およびローリングデプロイのトピックを参照してください。

拡張ヘルスレポートは、Elastic Load Balancing のヘルスチェック URL を適切に設定する必要があることについても指摘します。要求に対応するために環境がスケールアップすると、新しいインスタンスは、十分な数の ELB ヘルスチェックに合格するとすぐにリクエストの受け取りを開始します。ヘルスチェック URL が設定されていなければ、新しいインスタンスが TCP 接続を受け付けてからわずか 20 秒で開始することがあります。

ロードバランサーによってアプリケーションが正常であり、トラフィックを受け取っても問題がないと宣言されるまでにアプリケーションの起動が終了していない場合、大量のリクエストが失敗し、環境がヘルスチェックに合格しなくなります。アプリケーションによって処理されるパスをヒットするヘルスチェック URL が、この問題を防止できます。ヘルスチェック URL への GET リクエストが 200 ステータスコードを返すまで、ELB ヘルスチェックに合格しません。

Elastic Beanstalk の拡張ヘルスレポートの有効化

このトピックでは、拡張ヘルスレポートを有効にする方法について説明します。Elastic Beanstalk コンソール、EB CLI、および .ebextensions 設定を使用して、環境の拡張ヘルス機能を有効にする手順を示します。

最新のプラットフォームバージョンで作成された新しい環境には、拡張ヘルスレポートをサポートする AWS Elastic Beanstalk ヘルスエージェントが含まれています。Elastic Beanstalk コンソールまたは EB CLI を使用して環境を作成する場合は、拡張ヘルスレポートがデフォルトで有効になっています。設定ファイルを使用して、アプリケーションのソースコードでヘルスレポートの設定を変更することもできます。

拡張ヘルスレポートには、一連の標準のアクセス権限に加えて、インスタンスプロファイルとサービスロールが必要です。Elastic Beanstalk コンソールで環境を作成すると、Elastic Beanstalk によって自動的に必要なロールが作成されます。最初の環境を作成する手順については、「Elastic Beanstalk の使用を開始する方法について説明します。」を参照してください。

トピック

- Elastic Beanstalk コンソールを使用した拡張ヘルスレポートの有効化
- EB CLI を使用した拡張ヘルスレポートの有効化
- 設定ファイルを使用した拡張ヘルスレポートの有効化

Elastic Beanstalk コンソールを使用した拡張ヘルスレポートの有効化

Elastic Beanstalk コンソールを使用して実行中の環境で拡張ヘルスレポートを有効にするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [モニタリング] 設定カテゴリで、[編集] を選択します。
5. [ヘルスレポート] の [システム] で [Enhanced (拡張)] を選択します。

> **Note**
>
> サポートしていないプラットフォームまたはバージョンを使用している場合、拡張ヘルスレポートのオプションは表示されません。

6. ページの最下部で [適用] を選択し変更を保存します。

Elastic Beanstalk コンソールでは、バージョン 2 (v2) プラットフォーム設定で新しい環境を作成するときにデフォルトで拡張ヘルスレポートが有効になります。環境の作成中にヘルスレポートオプションを変更することによって、拡張ヘルスレポートを無効にすることができます。

Elastic Beanstalk コンソールを使用して環境を作成しているときに拡張ヘルスレポートを無効にするは

1. Elastic Beanstalk 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. 新しいアプリケーションを作成するか、既存のアプリケーションを選択します。
3. 環境を作成します。[新しい環境の作成] ページで、[環境の作成] を選択する前に、[さらにオプションを設定] を選択します。
4. [モニタリング] 設定カテゴリで、[編集] を選択します。
5. [Health reporting] の [System] で [Basic] を選択します。
6. [保存] を選択します。

EB CLI を使用した拡張ヘルスレポートの有効化

eb create コマンドを使用して新しい環境を作成すると、EB CLI では拡張ヘルスレポートがデフォルトで有効になり、デフォルトのインスタンスプロファイルとサービスロールが適用されます。

`--service-role` オプションを使用して、異なるサービスロールを名前で指定できます。

v2 プラットフォームバージョンで基本ヘルスレポートが使用されている環境を実行している場合、拡張ヘルスに切り替えるには、以下のステップを実行します。

EB CLI を使用して実行中の環境の拡張ヘルスレポートを有効にするには

1. eb config コマンドを使用して、設定ファイルをデフォルトのテキストエディタで開きます。

```
~/project$ eb config
```

2. 設定セクションで、`aws:elasticbeanstalk:environment` 名前空間を見つけます。`ServiceRole` の値が null ではなく、サービスロールの名前と一致していることを確認します。

```
  aws:elasticbeanstalk:environment:
    EnvironmentType: LoadBalanced
    ServiceRole: aws-elasticbeanstalk-service-role
```

3. `aws:elasticbeanstalk:healthreporting:system:` 名前空間で、`SystemType` の値を **enhanced** に変更します。

```
  aws:elasticbeanstalk:healthreporting:system:
    SystemType: enhanced
```

4. 設定ファイルを保存し、テキストエディタを閉じます。
5. EB CLI によって環境の更新が開始されて、設定の変更が適用されます。オペレーションの完了を待つか、Ctrl+C キーを押して安全に終了します。

```
~/project$ eb config
Printing Status:
INFO: Environment update is starting.
INFO: Health reporting type changed to ENHANCED.
INFO: Updating environment no-role-test's configuration settings.
```

設定ファイルを使用した拡張ヘルスレポートの有効化

ソースバンドルに設定ファイルを含めることで、拡張ヘルスレポートを有効にすることができます。以下の例では、拡張ヘルスレポートを有効にし、デフォルトのサービスとインスタンスプロファイルを環境に割り当てる、設定ファイルを示しています。

Example .ebextensions/enhanced-health.config

```
option_settings:
  aws:elasticbeanstalk:healthreporting:system:
    SystemType: enhanced
  aws:autoscaling:launchconfiguration:
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
  aws:elasticbeanstalk:environment:
    ServiceRole: aws-elasticbeanstalk-service-role
```

独自のインスタンスプロファイルやサービスロールを作成した場合は、強調表示されたテキストをそれらのロールの名前に置き換えます。

環境管理コンソールでの拡張ヘルスモニタリング

で拡張ヘルスレポートを有効にすると AWS Elastic Beanstalk、環境マネジメントコンソールで環境のヘルスをモニタリングできます。

トピック

- 環境の概要
- 環境の状態ページ
- モニタリングページ

環境の概要

環境の概要には、環境のヘルスステータスが表示され、ヘルスステータスの最新の変更に関する情報を提供するイベントが示されます。

環境の概要を表示するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。

現在の環境のヘルスに関する詳細情報を確認するには、[Causes (原因)] を選択して [ヘルス] ページを開きます。または、ナビゲーションペインで [ヘルス] を選択します。

環境の状態ページ

[Health (ヘルス)] ページには、環境とその環境内の各 Amazon EC2 インスタンスについてヘルスステータス、メトリクス、原因が表示されます。

> **Note**
>
> Elastic Beanstalk では、環境に対して拡張ヘルスモニタリングを有効にした場合にのみ、[Health] (ヘルス) ページが表示されます。

次の図は、Linux 環境の [ヘルス] ページを示しています。

次の図は、Windows 環境の [ヘルス] ページを示しています。CPU のメトリクスは、Linux 環境とは異なることに注意してください。

ページの上部には、環境インスタンスの合計数と、ステータスごとのインスタンス数が表示されます。特定のステータスのインスタンスのみを表示するには、[Filter By (フィルタ条件)] を選択してから [status (ステータス)] を選択します。

正常でないインスタンスを再起動/終了するには、[Instance Actions]、[Reboot]、[Terminate] の順に選択します。

Elastic Beanstalk では、[Health (ヘルス)] ページが 10 秒ごとに更新されます。このページでは、環境とインスタンスのヘルスに関する情報をレポートします。

環境内の各 Amazon EC2 インスタンスについて、このページには、インスタンスの ID とステータス、インスタンスが起動されてからの時間、インスタンスで実行された最新のデプロイの ID、インスタンスが処理したリクエストのレスポンスとレイテンシー、およびロードと CPU 使用率の情報が表示されます。[Overall (全体)] 行には、環境全体の平均レスポンスとレイテンシー情報が表示されます。

このページには、非常に横長のテーブルに多くの詳細が表示されます。一部の列を非表示にするには、
([Preferences (設定)]) を選択します。列名を選択または選択解除し、[Confirm (確認)] を選択します。

任意のインスタンスの [Instance ID (インスタンスの ID)] を選択すると、アベイラビリティーゾーンやインスタンスタイプなど、インスタンスの詳細情報が表示されます。

インスタンスの [Deployment ID (デプロイ ID)] を選択して、インスタンスへの前回のデプロイに関する情報を表示します。

デプロイに関する情報は以下のとおりです。

- Deployment ID (デプロイ ID)—デプロイの一意の識別子。デプロイ ID は 1 から始まり、新しいアプリケーションバージョンをデプロイするか、お客様の環境内のインスタンスで動作するソフトウェアやオペレーティングシステムに影響を与える構成設定を変更するたびに、1 ずつ増えます。
- Version (バージョン)—デプロイで使用されるアプリケーションのソースコードのバージョンラベル。
- Status (ステータス)—デプロイのステータス。`In Progress`、`Deployed`、または `Failed` になります。
- Time (時間)—進行中のデプロイの場合は、デプロイが開始した時間。完了したデプロイの場合は、デプロイが終了した時間。

環境で X-Ray 統合を有効にし、 AWS X-Ray SDK を使用してアプリケーションを計測する場合、ヘルスページは概要行に AWS X-Ray コンソールへのリンクを追加します。

リンクを選択すると、強調表示された統計に関連するトレースが AWS X-Ray コンソールに表示されます。

モニタリングページ

[Monitoring (モニタリング)] ページには、拡張ヘルスレポートシステムによって生成された Amazon CloudWatch カスタムメトリクスの統計サマリーとグラフが表示されます。このページにグラフと統計情報を追加する手順については、「AWS マネジメントコンソールでの環境の状態のモニタリング」を参照してください。

状態の色とステータス

拡張ヘルスレポートは、基本ヘルスレポートと同様に、インスタンスと環境全体の状態を 4 色を使って表します。また、拡張ヘルスレポートは、単一の単語で示される 7 つのヘルスステータスも表示します。これにより、環境の状態をより的確に把握できます。

インスタンスのステータスと環境ステータス

Elastic Beanstalk が環境のヘルスチェックを実行するたびに、拡張ヘルスレポートは、使用できるすべてのデータを分析することによって、環境内の各インスタンスの状態をチェックします。低いレベルのチェックに合格しなければ、Elastic Beanstalk はインスタンスの状態をダウングレードします。

Elastic Beanstalk は、環境全体のヘルス情報 (色、ステータス、および原因) を環境マネジメントコンソールに表示します。この情報は、EB CLI でも使用できます。個々のインスタンスのヘルスステータスと原因のメッセージは、10 秒ごとに更新され、EB CLI から eb health を使用してヘルスステータスを表示するときに確認できます。

Elastic Beanstalk は、インスタンスの状態の変化に基づいて環境の状態を評価しますが、環境のヘルスステータスをすぐに変更するわけではありません。インスタンスが 1 分間に 3 回以上ヘルスチェックに不合格になると、Elastic Beanstalk は環境の状態をダウングレードする場合があります。環境内のインスタンスの数と特定された問題によっては、1 つのインスタンスに異常があるだけで、Elastic Beanstalk が情報メッセージを表示する、または環境のヘルスステータスを緑色 (OK) から黄色 (Warning) または赤色 (Degraded または Severe) に変更できます。

OK (緑色)

このステータスは、以下の場合に表示されます。

- インスタンスはヘルスチェックに合格し、ヘルスエージェントは問題を報告していない。
- 環境内のほとんどのインスタンスがヘルスチェックに合格し、ヘルスエージェントは重大な問題を報告していない。
- インスタンスはヘルスチェックに合格し、リクエストを正常に完了している。

例: 環境がデプロイされたばかりであり、リクエストを正常に受け取っていない。5% のリクエストが 400 シリーズのエラーを返している。各インスタンスでデプロイが正常に完了した。

メッセージ (インスタンス): Application deployment completed 23 seconds ago and took 26 seconds。

Warning (黄色)

このステータスは、以下の場合に表示されます。

- ヘルスエージェントが、ある程度の数のリクエストが不合格であったこと、あるいはインスタンスまたは環境にその他の問題があることを報告している。
- インスタンスで進行中の操作に、非常に長い時間がかかっている。

例: 環境内の 1 つのインスタンスのステータスが Severe である。

メッセージ (環境): Impaired services on 1 out of 5 instances

Degraded (赤色)

このステータスが表示されるのは、ヘルスエージェントが、非常に多くのリクエストが不合格であったこと、あるいはインスタンスまたは環境にその他の問題があることを報告している場合です。

例: 環境が 5 つのインスタンスへのスケールアップを処理している。

メッセージ (環境): 4 active instances is below Auto Scaling group minimum size 5.

Severe (赤色)

このステータスが表示されるのは、ヘルスエージェントが、非常に多くのリクエストが不合格であったこと、あるいはインスタンスまたは環境にその他の問題があることを報告している場合です。

例: Elastic Beanstalk は、ロードバランサーにアクセスしてインスタンスの状態を取得することができません。

メッセージ (環境): ELB health is failing or not available for all instances。None of the instances are sending data。Unable to assume role "arn:aws:iam::123456789012:role/aws-elasticbeanstalk-service-role"。Verify that the role exists and is configured correctly。

メッセージ (インスタンス): Instance ELB health has not been available for 37 minutes。No data。Last seen 37 minutes ago。

Info (緑色)

このステータスは、以下の場合に表示されます。

- インスタンスで操作が進行中である。
- 環境内の複数のインスタンスで操作が進行中である。

例: 実行中のインスタンスに新しいアプリケーションバージョンがデプロイされている。

メッセージ (環境): Command is executing on 3 out of 5 instances。

メッセージ (インスタンス): Performing application deployment (running for 3 seconds)。

Pending (灰色)

このステータスが表示されるのは、コマンドタイムアウトの時間内でインスタンスでの操作が進行中のときです。

例: 環境を最近作成したばかりであり、インスタンスのブートストラップが行われている。

メッセージ: Performing initialization (running for 12 seconds)。

Unknown (灰色)

このステータスが表示されるのは、Elastic Beanstalk とヘルスエージェントが、インスタンスのデータの量が不足していることを報告したときです。

例: データを受け取っていない。

停止 (グレー)

このステータスが表示されるのは、Elastic Beanstalk が環境のヘルス状態のモニタリングを停止したときです。環境は正常に動作しない可能性があります。一部の重大なヘルス条件が長期間存在する場合、Elastic Beanstalk により環境が停止状態になります。

例: Elastic Beanstalk が環境のサービスロールにアクセスできない。

例: 環境に対して Elastic Beanstalk によって作成された Auto Scaling グループが削除された。

メッセージ: 環境のヘルスステータスが [OK] から [重大] に移行されました。インスタンスはありません。Auto Scaling グループの希望する容量が 1 に設定されます。

インスタンスメトリクス

インスタンスメトリクスは、環境にあるインスタンスの健全性に関する情報を提供します。Elastic Beanstalk ヘルスエージェントは、各インスタンスで実行されます。ヘルスエージェントはインスタンスに関するメトリクスを収集し、中継された Elastic Beanstalk はそのメトリクスを分析して環境内のインスタンスの状態を特定します。

オンインスタンス Elastic Beanstalk ヘルスエージェントは、ウェブサーバーログとオペレーティングシステムからインスタンスに関するメトリクスを収集します。Linux ベースのプラットフォームで

ウェブサーバー情報を取得するために、Elastic Beanstalk はウェブサーバーのログを読み取り、解析します。Windows Server プラットフォームでは、Elastic Beanstalk はこの情報を IIS ウェブサーバーから直接受け取ります。ウェブサーバーは、着信 HTTP リクエストに関する情報 (届いたリクエストの数、エラーとなった数、解決までの時間) を提供します。オペレーティング システムはインスタンスのリソース状態についてのスナップショット情報を提供します。プロセスタイプごとの CPU 負荷および配信経過時間。

ヘルスエージェントはウェブサーバーとオペレーティングシステムのメトリクスを収集し、10 秒ごとに Elastic Beanstalk に中継します。Elastic Beanstalk は中継されたデータを分析し、その結果を使用して、各インスタンスと環境のヘルスステータスを更新します。

トピック

- ウェブサーバーのメトリクス
- オペレーティングシステムのメトリクス
- Windows サーバー上の IIS でのウェブサーバーメトリクスキャプチャ

ウェブサーバーのメトリクス

Linux ベースのプラットフォームで、Elastic Beanstalk ヘルスエージェントは、ウェブコンテナまたは環境内の各インスタンスでリクエストを処理するサーバーによって生成されたログからウェブサーバーメトリクスを読み取ります。Elastic Beanstalk プラットフォームは、人間が読み取れる形式と機械による読み取りが可能な形式の 2 つのログを生成するように設定されています。機械による読み取りが可能なログは、ヘルスエージェントによって 10 秒ごとに Elastic Beanstalk に中継されます。

Elastic Beanstalk で使用されるログ形式の詳細については、「拡張ヘルスログ形式」を参照してください。

Windows Server プラットフォームでは、Elastic Beanstalk は IIS ウェブサーバーのリクエストパイプラインにモジュールを追加し、HTTP リクエスト時間と応答コードに関するメトリクスを取得します。モジュールは、高性能のプロセス間通信 (IPC) チャネルを使用して、これらのメトリクスをインスタンス上のヘルスエージェントに送信します。実装の詳細については、「Windows サーバー上の IIS でのウェブサーバーメトリクスキャプチャ」を参照してください。

報告されたウェブサーバーのメトリクス

`RequestCount`

直前の 10 秒間にウェブサーバーによって処理されたリクエストの 1 秒あたりの数。EB CLI と 環境の状態ページ に表示される平均 `r/sec` (1 秒ごとのリクエスト) として表示されます。

`Status2xx`, `Status3xx`, `Status4xx`, `Status5xx`

直前の 10 秒間に各タイプのステータスコードが返されたリクエストの数。たとえば、正常なリクエストには 200 OK、リダイレクトには 301 が返され、アプリケーション内のどのリソースとも一致しない URL が入力された場合は 404 が返されます。

EB CLI と 環境の状態ページ は、インスタンスへのリクエスト未処理数、そして環境内の総体的なリクエストのパーセンテージとしてこれらのメトリクスを示します。

`p99.9`, `p99`, `p95`, `p90`, `p85`, `p75`, `p50`, `p10`

最近 10 秒間で最も遅かったリクエストの x パーセントの平均レイテンシー。x はこの数値と 100 との差異です。たとえば、`p99 1.403` は、直前の 10 秒間に応答が返るのが最も遅かったリクエストの 1% の平均レイテンシーが 1.403 秒であったことを示します。

オペレーティングシステムのメトリクス

Elastic Beanstalk ヘルスエージェントは、以下のオペレーティングシステムメトリクスを報告します。Elastic Beanstalk は、これらのメトリクスを使用して、継続的に重い負荷がかかっているインスタンスを識別します。メトリクスはオペレーティングシステムによって異なります。

報告されているオペレーティングシステム (Linux)

`Running`

インスタンスが起動してから経過した時間。

`Load 1`, `Load 5`

直前の 1 分間と 5 分間の平均負荷。この期間に実行されていたプロセスの平均数を小数値で示します。表示された数が使用可能な vCPU (スレッド) の数よりも多い場合、余りは待機中だったプロセスの平均数です。

たとえば、インスタンスタイプが 4 vCPU であり、負荷が 4.5 である場合、その期間において、平均で .5 のプロセスが待機していたことになり、その期間の 50% にわたって 1 つのプロセスが待機していたことを意味します。

`User %`, `Nice %`, `System %`, `Idle %`, `I/O Wait %`

過去 10 秒間に CPU が各状態で費やした時間のパーセンテージ。

報告されているオペレーティングシステム (Windows)

`Running`

インスタンスが起動してから経過した時間。

`% User Time`, `% Privileged Time`, `% Idle Time`

過去 10 秒間に CPU が各状態で費やした時間のパーセンテージ。

Windows サーバー上の IIS でのウェブサーバーメトリクスキャプチャ

Windows Server プラットフォームでは、Elastic Beanstalk は IIS ウェブサーバーのリクエストパイプラインにモジュールを追加し、HTTP リクエスト時間と応答コードに関するメトリクスを取得します。モジュールは、高性能のプロセス間通信 (IPC) チャネルを使用して、これらのメトリクスをインスタンス上のヘルスエージェントに送信します。ヘルスエージェントは、これらのメトリクスを集計し、オペレーティングシステムメトリクスと組み合わせて、Elastic Beanstalk サービスに送信します。

実装の詳細

IIS からメトリクスを取得するために、Elastic Beanstalk はマネージド型 `IHttpModule` を実装して、`BeginRequest` および `EndRequest` イベントをサブスクライブします。これにより、モジュールのレイテンシーと応答コード HTTP リクエストを報告するウェブリクエストはすべて IIS によって処理されます。モジュールを IIS のリクエストパイプラインに追加するために、Elastic Beanstalk は IIS の設定ファイル、`%windir%\System32\inetsrv\config\applicationHost.config` の `<modules>` セクションにモジュールを登録します。

IIS の Elastic Beanstalk モジュールは、キャプチャされたウェブリクエストモジュールメトリクスを、`HealthD` という名前の Windows サービスであるインスタンス上のヘルスエージェントに送信します。このデータを送信するために、モジュールは `NetNamedPipeBinding` を使用します。これは、マシン上の通信に最適化された安全で信頼性の高いバインディングを提供します。

環境の拡張ヘルスルールの設定

AWS Elastic Beanstalk 拡張ヘルスレポートは、環境のヘルスを判断するために一連のルールに依存します。これらのルールの一部は、特定のアプリケーションに適していない場合があります。一般的な例をいくつか以下に示します。

- クライアント側のテストツールを使用する。この場合、HTTP クライアント (4xx) エラーが頻発することが予想されます。

- AWS WAF を環境の Application Load Balancer と併用して不要な着信トラフィックをブロックします。この場合、Application Load Balancer は着信メッセージを拒否するたびに HTTP 403 を返します。

デフォルトでは、Elastic Beanstalk はアプリケーションのすべての HTTP 4xx エラーを反映して環境のヘルスを判断します。これにより、環境のヘルスステータスが [OK] から [Warning] (警告)、[Degraded] (低下)、または [Severe] (重大) へと、エラー率に応じて変更されます。上のような例に正しく対処するために、Elastic Beanstalk ではいくつかの拡張ヘルスルールを設定できます。環境のインスタンスでアプリケーションの HTTP 4xx エラーを無視するか、環境のロードバランサーから返された HTTP 4xx エラーを無視するかを選択できます。このトピックでは、これらの設定変更を行う方法について説明します。

> **Note**
>
> 現在利用できる拡張ヘルスルールのカスタマイズは以上のみです。4xx 以外の HTTP エラーを無視するように拡張ヘルスを設定することはできません。

Elastic Beanstalk コンソールを使用した拡張ヘルスレポートの設定

Elastic Beanstalk コンソールを使用して環境で拡張ヘルスルールを設定できます。

Elastic Beanstalk コンソールを使用して HTTP 4xx ステータスコードのチェックを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [モニタリング] 設定カテゴリで、[編集] を選択します。
5. [Health monitoring rule customization] で、該当する [Ignore] オプションを有効または無効にします。
6. ページの最下部で [適用] を選択し変更を保存します。

EB CLI を使用して拡張ヘルスルールを設定する

EB CLI を使用すると、環境の設定をローカルに保存し、拡張ヘルスルールを設定するエントリを追加してから、その設定を Elastic Beanstalk にアップロードすることによって、拡張ヘルスルールを設定できます。保存した設定は、環境を作成する前または作成した後に環境に適用できます。

EB CLI および保存済みの設定を使用して HTTP 4xx ステータスコードのチェックを設定するには

1. eb init でプロジェクトフォルダを初期化します。
2. eb create コマンドを実行して、環境を作成します。
3. eb config save コマンドを実行して、設定テンプレートをローカルに保存します。次の例では、`--cfg` オプションを使用して、設定の名前が指定されています。

```
$ eb config save --cfg 01-base-state
Configuration saved at: ~/project/.elasticbeanstalk/saved_configs/01-base-
state.cfg.yml
```

4. 保存した設定ファイルをテキストエディタで開きます。
5. `OptionSettings` > `aws:elasticbeanstalk:healthreporting:system:` で、設定する各拡張ヘルスルールを一覧表示する `ConfigDocument` キーを追加します。次の `ConfigDocument` では、ロードバランサーの HTTP 4xx コードのチェックを有効にしたままで、アプリケーションの HTTP 4xx ステータスコードのチェックを無効にします。

```
OptionSettings:
  ...
  aws:elasticbeanstalk:healthreporting:system:
    ConfigDocument:
      Rules:
        Environment:
          Application:
            ApplicationRequests4xx:
              Enabled: false
          ELB:
            ELBRequests4xx:
              Enabled: true
      Version: 1
    SystemType: enhanced
...
```

> **Note**
>
> 同じ ConfigDocument オプション設定で、Rules と CloudWatchMetrics を組み合わせることができます。CloudWatchMetrics については、「環境の Amazon CloudWatch カスタムメトリクスの発行」で説明しています。
> 以前に CloudWatchMetrics を有効にしている場合、eb config save コマンドを使用して取得した設定ファイルには、すでに ConfigDocument キーが CloudWatchMetrics セクションにあります。削除しないでください。同じ ConfigDocument オプション値に Rules セクションを追加します。

6. 設定ファイルを保存し、テキストエディタを閉じます。この例では、更新した設定ファイルは、ダウンロードした設定ファイルとは異なる名前 (02-cloudwatch-enabled.cfg.yml) で保存されています。このファイルがアップロードされると、別の保存済み設定が作成されます。ダウンロードしたファイル同じ名前を使用すると、新しいキーペアを作成せずに既存の設定を上書きできます。
7. eb config put コマンドを使用して、更新した設定ファイルを Elastic Beanstalk にアップロードします。

```
$ eb config put 02-cloudwatch-enabled
```

   保存した設定に対して eb config get コマンドと put コマンドを使用するときは、ファイル名の拡張子を含めないでください。
8. 実行中の環境に、保存済みの設定を適用します。

```
$ eb config --cfg 02-cloudwatch-enabled
```

   --cfg オプションは、環境に適用される名前付き設定ファイルを指定します。設定ファイルはローカルまたは Elastic Beanstalk に保存できます。指定した名前を持つ設定ファイルが両方の場所に存在する場合、EB CLI はローカルファイルを使用します。

設定ドキュメントを使用して拡張ヘルスルールを設定する

拡張ヘルスルールの設定 (config) ドキュメントは、設定するルールを一覧表示した JSON ドキュメントです。

次の例に示す設定ドキュメントでは、アプリケーションの HTTP 4xx ステータスコードのチェックを無効にし、ロードバランサーの HTTP 4xx ステータスコードのチェックを有効にします。

```
{
  "Rules": {
    "Environment": {
      "Application": {
        "ApplicationRequests4xx": {
          "Enabled": false
        }
      },
      "ELB": {
        "ELBRequests4xx": {
          "Enabled": true
        }
      }
    }
  },
  "Version": 1
}
```

の場合 AWS CLI、 ドキュメントをオプション設定引数の`Value`キーの値として渡します。これは、それ自体が JSON オブジェクトです。この場合、埋め込まれているドキュメントの引用符はエスケープする必要があります。次のコマンドは、設定が有効であるかどうかを確認します。

```
$ aws elasticbeanstalk validate-configuration-settings --application-name my-app --
environment-name my-env --option-settings '[
    {
        "Namespace": "aws:elasticbeanstalk:healthreporting:system",
        "OptionName": "ConfigDocument",
        "Value": "{\"Rules\": { \"Environment\": { \"Application\":
 { \"ApplicationRequests4xx\": { \"Enabled\": false } }, \"ELB\": { \"ELBRequests4xx\":
 {\"Enabled\": true } } } }, \"Version\": 1 }"
    }
]'
```

YAML の `.ebextensions` 設定ファイルの場合は、JSON ドキュメントをそのまま提供できます。

```
  option_settings:
    - namespace: aws:elasticbeanstalk:healthreporting:system
      option_name: ConfigDocument
```

```
      value: {
  "Rules": {
    "Environment": {
      "Application": {
        "ApplicationRequests4xx": {
          "Enabled": false
        }
      },
      "ELB": {
        "ELBRequests4xx": {
          "Enabled": true
        }
      }
    }
  },
  "Version": 1
}
```

環境の Amazon CloudWatch カスタムメトリクスの発行

AWS Elastic Beanstalk 拡張ヘルスレポートによって収集されたデータをカスタムメトリクスとして Amazon CloudWatch に公開できます。CloudWatch にメトリクスをパブリッシュすることにより、時間の経過に伴うアプリケーションのパフォーマンスの変化をモニタリングできるほか、リソースの使用状況やリクエストのレイテンシーが負荷に応じてどのようにスケーリングするかを追跡することによって、発生する可能性のある問題を特定できます。

また、CloudWatch にメトリクスをパブリッシュすることにより、モニタリンググラフとアラームでメトリクスを使用できます。無料のメトリクスである EnvironmentHealth は、拡張ヘルスレポートを使用するとき、自動的に有効になります。EnvironmentHealth 以外のカスタムメトリクスを使用する場合、CloudWatch の標準料金が課金されます。

環境の CloudWatch カスタムメトリクスをパブリッシュするには、まず環境で拡張ヘルスレポートを有効にする必要があります。手順については「Elastic Beanstalk の拡張ヘルスレポートの有効化」を参照してください。

トピック

- 拡張ヘルスレポートのメトリクス
- Elastic Beanstalk コンソールを使用した CloudWatch メトリクスの設定
- EB CLI を使用した CloudWatch カスタムメトリクスの設定
- カスタムメトリクス設定ドキュメントの提供

拡張ヘルスレポートのメトリクス

環境で拡張ヘルスレポートを有効にすると、拡張ヘルスレポートシステムが CloudWatch カスタムメトリクスの 1 つである EnvironmentHealth を自動的にパブリッシュします。追加のメトリクスを CloudWatch にパブリッシュするには、Elastic Beanstalk コンソール、EB CLI、または .ebextensions を使用して、これらのメトリクスで環境を設定します。

環境から、次の拡張ヘルスメトリクスを CloudWatch にパブリッシュすることができます。

使用可能なメトリクス - すべてのプラットフォーム

`EnvironmentHealth`

環境のみが対象。他のメトリクスを設定していなければ、拡張ヘルスレポートシステムからパブリッシュされる唯一の CloudWatch メトリクスです。環境の状態は、7 種類のステータスのいずれかで表されます。CloudWatch コンソールでは、これらのステータスは以下の値にマッピングされます。

- 0 – OK
- 1 – Info
- 5 – Unknown
- 10 – No data
- 15 – Warning
- 20 – Degraded
- 25 – Severe

`InstancesSevere`, `InstancesDegraded`, `InstancesWarning`, `InstancesInfo`, `InstancesOk`, `InstancesPending`, `InstancesUnknown`, `InstancesNoData`

環境のみが対象。これらのメトリクスは、各ヘルスステータスにある環境内のインスタンスの数を示します。`InstancesNoData` は、データを受け取っていないインスタンスの数を示します (該当する場合)。

`ApplicationRequestsTotal`, `ApplicationRequests5xx`, `ApplicationRequests4xx`, `ApplicationRequests3xx`, `ApplicationRequests2xx`

インスタンスと環境が対象。インスタンスまたは環境で完了したリクエストの総数と、各ステータスコードカテゴリで完了したリクエストの数を示します。

`ApplicationLatencyP10`, `ApplicationLatencyP50`, `ApplicationLatencyP75`, `ApplicationLatencyP85`, `ApplicationLatencyP90`, `ApplicationLatencyP95`, `ApplicationLatencyP99`, `ApplicationLatencyP99.9`

インスタンスと環境が対象。リクエストのうち、早い方から x パーセントの完了にかかった平均時間を直ちに示します。

`InstanceHealth`

インスタンスのみが対象。インスタンスの現在のヘルスステータスを示します。インスタンスの状態は、7 種類のステータスのいずれかで表されます。CloudWatch コンソールでは、これらのステータスは以下の値にマッピングされます。

- 0 – OK
- 1 – Info
- 5 – Unknown
- 10 – No data
- 15 – Warning
- 20 – Degraded
- 25 – Severe

使用可能なメトリクス - Linux

`CPUIrq`, `CPUIdle`, `CPUUser`, `CPUSystem`, `CPUSoftirq`, `CPUIowait`, `CPUNice`

インスタンスのみが対象。過去 1 分間に CPU が各状態で消費した時間の割合を示します。

`LoadAverage1min`

インスタンスのみが対象。インスタンスに関する過去 1 分間の CPU 負荷の平均値。

`RootFilesystemUtil`

インスタンスのみが対象。使用中のディスク容量の割合を示します。

利用可能なメトリクス - Windows

`CPUIdle`, `CPUUser`, `CPUPrivileged`

インスタンスのみが対象。過去 1 分間に CPU が各状態で消費した時間の割合を示します。

Elastic Beanstalk コンソールを使用した CloudWatch メトリクスの設定

Elastic Beanstalk コンソールを使用して、拡張ヘルスレポートのメトリクスを CloudWatch にパブリッシュし、モニタリンググラフとアラームで使用できるように環境を設定できます。

Elastic Beanstalk コンソールで CloudWatch カスタムメトリクスを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [モニタリング] 設定カテゴリで、[編集] を選択します。
5. [Health reporting (ヘルプレポート)] で、CloudWatch に公開するインスタンスと環境のメトリクスを選択します。複数のメトリクスを選択するには、Ctrl キーを押して選択します。
6. ページの最下部で [適用] を選択し変更を保存します。

CloudWatch カスタムメトリクスを有効にすると、[Monitoring (モニタリング)] ページで使用できるメトリクスのリストにこれらのメトリクスが追加されます。

EB CLI を使用した CloudWatch カスタムメトリクスの設定

EB CLI を使用すると、環境の設定をローカルに保存し、パブリッシュするメトリクスを定義するエントリを追加してから、その設定を Elastic Beanstalk にアップロードすることによって、カスタムメトリクスを設定できます。保存した設定は、環境を作成する前または作成した後に環境に適用できます。

EB CLI と保存した設定を使用して CloudWatch カスタムメトリクスを設定するには

1. eb init でプロジェクトフォルダを初期化します。
2. eb create コマンドを実行して、環境を作成します。
3. eb config save コマンドを実行して、設定テンプレートをローカルに保存します。次の例では、`--cfg` オプションを使用して、設定の名前が指定されています。

```
$ eb config save --cfg 01-base-state
Configuration saved at: ~/project/.elasticbeanstalk/saved_configs/01-base-
state.cfg.yml
```

4. 保存した設定ファイルをテキストエディタで開きます。

5. `OptionSettings` > `aws:elasticbeanstalk:healthreporting:system:` で、CloudWatch メトリクスを個別に有効にする `ConfigDocument` キーを追加します。たとえば、次に示す `ConfigDocument` は、環境レベルで `ApplicationRequests5xx` メトリクスと `ApplicationRequests4xx` メトリクスをパブリッシュし、インスタンスレベルで `ApplicationRequestsTotal` メトリクスをパブリッシュします。

```
OptionSettings:
  ...
  aws:elasticbeanstalk:healthreporting:system:
    ConfigDocument:
      CloudWatchMetrics:
        Environment:
          ApplicationRequests5xx: 60
          ApplicationRequests4xx: 60
        Instance:
          ApplicationRequestsTotal: 60
      Version: 1
    SystemType: enhanced
...
```

この例では、60 は測定間隔の秒数を示しています。これは、現在サポートされている唯一の値です。

> **Note**
>
> 同じ `ConfigDocument` オプション設定で、`CloudWatchMetrics` と `Rules` を組み合わせることができます。`Rules` については、「環境の拡張ヘルスルールの設定」で説明しています。
> 以前に `Rules` を使用して拡張ヘルスルールを設定している場合、eb config save コマンドを使用して取得される設定ファイルには、既に `ConfigDocument` キーが `Rules` セクションにあります。削除しないでください。同じ `ConfigDocument` オプション値に `CloudWatchMetrics` セクションを追加します。

6. 設定ファイルを保存し、テキストエディタを閉じます。この例では、更新した設定ファイルは、ダウンロードした設定ファイルとは異なる名前 (`02-cloudwatch-enabled.cfg.yml`) で保存されています。このファイルがアップロードされると、別の保存済み設定が作成されます。ダウンロードしたファイル同じ名前を使用すると、新しいキーペアを作成せずに既存の設定を上書きできます。

7. eb config put コマンドを使用して、更新した設定ファイルを Elastic Beanstalk にアップロードします。

```
$ eb config put 02-cloudwatch-enabled
```

保存した設定に対して eb config get コマンドと put コマンドを使用するときは、ファイル拡張子を含めないでください。

8. 実行中の環境に、保存済みの設定を適用します。

```
$ eb config --cfg 02-cloudwatch-enabled
```

--cfg オプションは、環境に適用される名前付き設定ファイルを指定します。設定ファイルはローカルまたは Elastic Beanstalk に保存できます。指定した名前を持つ設定ファイルが両方の場所に存在する場合、EB CLI はローカルファイルを使用します。

カスタムメトリクス設定ドキュメントの提供

Amazon CloudWatch カスタムメトリクスの設定 (config) ドキュメントは、環境レベルとインスタンスレベルでパブリッシュするメトリクスを一覧表示する JSON ドキュメントです。次の例は、すべての利用可能なカスタムメトリクスを Linux で有効にする設定ドキュメントを示しています。

```
{
  "CloudWatchMetrics": {
    "Environment": {
      "ApplicationLatencyP99.9": 60,
      "InstancesSevere": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "InstancesUnknown": 60,
      "ApplicationLatencyP85": 60,
      "InstancesInfo": 60,
      "ApplicationRequests2xx": 60,
      "InstancesDegraded": 60,
      "InstancesWarning": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "InstancesNoData": 60,
      "InstancesPending": 60,
```

```
      "ApplicationLatencyP10": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "InstancesOk": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60
    },
    "Instance": {
      "ApplicationLatencyP99.9": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "ApplicationLatencyP85": 60,
      "CPUUser": 60,
      "ApplicationRequests2xx": 60,
      "CPUIdle": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "RootFilesystemUtil": 60,
      "LoadAverage1min": 60,
      "CPUIrq": 60,
      "CPUNice": 60,
      "CPUIowait": 60,
      "ApplicationLatencyP10": 60,
      "LoadAverage5min": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "CPUSystem": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60,
      "InstanceHealth": 60,
      "CPUSoftirq": 60
    }
  },
  "Version": 1
}
```

の場合 AWS CLI、 ドキュメントをオプション設定引数の`Value`キーの値として渡します。これは、それ自体が JSON オブジェクトです。この場合、埋め込まれているドキュメントの引用符はエスケープする必要があります。

```
$ aws elasticbeanstalk validate-configuration-settings --application-name my-app --
environment-name my-env --option-settings '[
```

```
    {
        "Namespace": "aws:elasticbeanstalk:healthreporting:system",
        "OptionName": "ConfigDocument",
        "Value": "{\"CloudWatchMetrics\": {\"Environment\":
 {\"ApplicationLatencyP99.9\": 60,\"InstancesSevere\": 60,\"ApplicationLatencyP90\":
 60,\"ApplicationLatencyP99\": 60,\"ApplicationLatencyP95\": 60,\"InstancesUnknown
\": 60,\"ApplicationLatencyP85\": 60,\"InstancesInfo\": 60,\"ApplicationRequests2xx
\": 60,\"InstancesDegraded\": 60,\"InstancesWarning\": 60,\"ApplicationLatencyP50\":
 60,\"ApplicationRequestsTotal\": 60,\"InstancesNoData\": 60,\"InstancesPending
\": 60,\"ApplicationLatencyP10\": 60,\"ApplicationRequests5xx\": 60,
\"ApplicationLatencyP75\": 60,\"InstancesOk\": 60,\"ApplicationRequests3xx\": 60,
\"ApplicationRequests4xx\": 60},\"Instance\": {\"ApplicationLatencyP99.9\": 60,
\"ApplicationLatencyP90\": 60,\"ApplicationLatencyP99\": 60,\"ApplicationLatencyP95\":
 60,\"ApplicationLatencyP85\": 60,\"CPUUser\": 60,\"ApplicationRequests2xx\":
 60,\"CPUIdle\": 60,\"ApplicationLatencyP50\": 60,\"ApplicationRequestsTotal\":
 60,\"RootFilesystemUtil\": 60,\"LoadAverage1min\": 60,\"CPUIrq\": 60,\"CPUNice
\": 60,\"CPUIowait\": 60,\"ApplicationLatencyP10\": 60,\"LoadAverage5min\": 60,
\"ApplicationRequests5xx\": 60,\"ApplicationLatencyP75\": 60,\"CPUSystem\": 60,
\"ApplicationRequests3xx\": 60,\"ApplicationRequests4xx\": 60,\"InstanceHealth\": 60,
\"CPUSoftirq\": 60}},\"Version\": 1}"
    }
]'
```

YAML の `.ebextensions` 設定ファイルの場合は、JSON ドキュメントをそのまま提供できます。

```
  option_settings:
    - namespace: aws:elasticbeanstalk:healthreporting:system
      option_name: ConfigDocument
      value: {
  "CloudWatchMetrics": {
    "Environment": {
      "ApplicationLatencyP99.9": 60,
      "InstancesSevere": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "InstancesUnknown": 60,
      "ApplicationLatencyP85": 60,
      "InstancesInfo": 60,
      "ApplicationRequests2xx": 60,
      "InstancesDegraded": 60,
      "InstancesWarning": 60,
      "ApplicationLatencyP50": 60,
```

```
      "ApplicationRequestsTotal": 60,
      "InstancesNoData": 60,
      "InstancesPending": 60,
      "ApplicationLatencyP10": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "InstancesOk": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60
    },
    "Instance": {
      "ApplicationLatencyP99.9": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "ApplicationLatencyP85": 60,
      "CPUUser": 60,
      "ApplicationRequests2xx": 60,
      "CPUIdle": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "RootFilesystemUtil": 60,
      "LoadAverage1min": 60,
      "CPUIrq": 60,
      "CPUNice": 60,
      "CPUIowait": 60,
      "ApplicationLatencyP10": 60,
      "LoadAverage5min": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "CPUSystem": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60,
      "InstanceHealth": 60,
      "CPUSoftirq": 60
    }
  },
  "Version": 1
}
```

Elastic Beanstalk API での拡張ヘルスレポートの使用

AWS Elastic Beanstalk 拡張ヘルスレポートにはロールとソリューションスタックの要件があるため、拡張ヘルスレポートのリリース前に使用したスクリプトとコードは、使用する前に更新する必要があります。下位互換性を維持するため、拡張ヘルスレポートは、Elastic Beanstalk API を使用して環境を作成するとき、デフォルトでは有効になりません。

環境のサービスロール、インスタンスプロファイル、および Amazon CloudWatch 設定オプションを設定することによって、拡張ヘルスレポートを設定します。これは、`.ebextensions` フォルダーで設定オプションを設定する、保存されている設定を使用する、または `create-environment` 呼び出しの `option-settings` パラメータで直接設定オプションを設定することによって行うことができます。

API、SDKs、または AWS コマンドラインインターフェイス (CLI) を使用して、拡張ヘルスをサポートする環境を作成するには、以下を実行する必要があります。

- 適切なアクセス権限を含むサービスロールとインスタンスプロファイルを作成する
- 新しいプラットフォームのバージョンで新しい環境を作成する
- ヘルスシステムのタイプ、インスタンスプロファイル、サービスロールの設定オプションを設定する

`aws:elasticbeanstalk:healthreporting:system`、`aws:autoscaling:launchconfiguration`、および `aws:elasticbeanstalk:environment` の 3 つの名前空間で以下の設定オプションを使用して、拡張ヘルスレポートの環境を設定します。

拡張ヘルスの設定オプション

SystemType

名前空間: `aws:elasticbeanstalk:healthreporting:system`

拡張ヘルスレポートを有効にするには、「**`enhanced`**」に設定します。

IamInstanceProfile

名前空間: `aws:autoscaling:launchconfiguration`

Elastic Beanstalk 用に設定されたインスタンスプロファイルの名前に設定します。

ServiceRole

名前空間: `aws:elasticbeanstalk:environment`

Elastic Beanstalk 用に設定されたサービスロールの名前に設定します。

ConfigDocument (オプション)

名前空間: `aws:elasticbeanstalk:healthreporting:system`

インスタンスと CloudWatch にパブリッシュする環境メトリクスを定義する JSON ドキュメント。次に例を示します。

```
{
  "CloudWatchMetrics":
    {
    "Environment":
      {
      "ApplicationLatencyP99.9":60,
      "InstancesSevere":60
      }
    "Instance":
      {
      "ApplicationLatencyP85":60,
      "CPUUser": 60
      }
    }
  "Version":1
}
```

> **Note**
>
> 設定ドキュメントでは、Elastic Beanstalk への提供方法に応じて、引用符のエスケープなどの特別なフォーマットが必要になることがあります。例については、「カスタムメトリクス設定ドキュメントの提供」を参照してください。

拡張ヘルスログ形式

AWS Elastic Beanstalk プラットフォームは、カスタムウェブサーバーのログ形式を使用して、HTTP リクエストに関する情報を効率的に拡張ヘルスレポートシステムに中継します。システム

は ログを分析し、問題を特定して、それに基づいてインスタンスと環境ヘルスを設定します。対象環境でウェブサーバープロキシを無効にし、ウェブコンテナから直接リクエストを処理する場合でも、Elastic Beanstalk ヘルスエージェントによって使用される場所と形式でログを出力するようにサーバーを設定することで、拡張ヘルスレポートを最大限に活用できます。

> **Note**
>
> このページの情報は、Linux ベースのプラットフォームにのみ関係します。Windows Server プラットフォームでは、Elastic Beanstalk は HTTP リクエストに関する情報を IIS ウェブサーバーから直接受け取ります。詳細については、「Windows サーバー上の IIS でのウェブサーバーメトリクスキャプチャ」を参照してください。

ウェブサーバーログ設定

Elastic Beanstalk プラットフォームは、HTTP リクエストに関する情報を含む 2 つのログを出力するように設定されています。最初のログは、詳細形式であり、リクエストに関する詳細な情報 (リクエスタのユーザーエージェント情報や人間が読み取れる形式のタイムスタンプなど) を提供します。

/var/log/nginx/access.log

以下の例は、Ruby ウェブサーバー環境で実行されている nginx プロキシのものですが、その形式は Apache のものに似ています。

```
172.31.24.3 - - [23/Jul/2015:00:21:20 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:21 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
```

2 つ目のログは、簡易形式です。これは、拡張ヘルスレポートに関連する情報のみを提供します。このログは、healthd という名前のサブフォルダに出力され、1 時間ごとにローテーションされます。古いログは、ローテーション直後に削除されます。

/var/log/nginx/healthd/application.log.2015-07-23-00

次の例は、機械による読み取りが可能な形式のログを示しています。

```
1437609879.311"/"200"0.083"0.083"177.72.242.17
1437609879.874"/"200"0.347"0.347"177.72.242.17
1437609880.006"/bad/path"404"0.001"0.001"177.72.242.17
1437609880.058"/"200"0.530"0.530"177.72.242.17
1437609880.928"/bad/path"404"0.001"0.001"177.72.242.17
```

拡張ヘルスログ形式には、以下の情報が含まれます。

- リクエストの時間 (Unix 時間)。
- リクエストのパス
- 結果に対応する HTTP ステータスコード
- リクエスト時間
- アップストリーム時間
- X-Forwarded-For HTTP ヘッダー

nginx プロキシでは、時間は小数点以下 3 桁の浮動小数点の秒数で表示されます。Apache では、整数で表したミリ秒数が使用されます。

> **Note**
>
> 次のような警告 (DATE-TIME は日付と時刻) がログファイルに表示される場合で、マルチコンテナ Docker 環境などでカスタムプロキシを使用しているときは、healthd がログファイルを読み込むことができるように、.ebextension を使用して環境を設定する必要があります。
>
> ```
> W, [DATE-TIME #1922] WARN -- : log file "/var/log/nginx/healthd/
> application.log.DATE-TIME" does not exist
> ```
>
> .ebextension はマルチコンテナ Docker のサンプルから開始できます。

/etc/nginx/conf.d/webapp_healthd.conf

この例は、healthd ログ形式が強調表示されている nginx のログ設定を示しています。

```
upstream my_app {
  server unix:///var/run/puma/my_app.sock;
}

log_format healthd '$msec"$uri"'
                '$status"$request_time"$upstream_response_time"'
                '$http_x_forwarded_for';

server {
  listen 80;
  server_name _ localhost; # need to listen to localhost for worker tier

  if ($time_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2})") {
    set $year $1;
    set $month $2;
    set $day $3;
    set $hour $4;
  }

  access_log  /var/log/nginx/access.log  main;
  access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour healthd;

  location / {
    proxy_pass http://my_app; # match the name of upstream directive which is defined
 above
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  }

  location /assets {
    alias /var/app/current/public/assets;
    gzip_static on;
    gzip on;
    expires max;
    add_header Cache-Control public;
  }

  location /public {
    alias /var/app/current/public;
    gzip_static on;
```

```
    gzip on;
    expires max;
    add_header Cache-Control public;
  }
}
```

/etc/httpd/conf.d/healthd.conf

次の例は、Apache のログ設定を示します。

```
LogFormat "%{%s}t\"%U\"%s\"%D\"%D\"%{X-Forwarded-For}i" healthd
CustomLog "|/usr/sbin/rotatelogs /var/log/httpd/healthd/application.log.%Y-%m-%d-%H
 3600" healthd
```

拡張ヘルスレポート用のログの生成

ヘルスエージェントにログを提供するには、以下の操作を行う必要があります。

- 前のセクションで示したように、正しい形式でログを出力する
- ログを `/var/log/nginx/healthd/` に出力する
- `application.log.$year-$month-$day-$hour`
- ログを 1 時間に 1 回ローテーションさせる
- ログは切り捨てないでください

通知とトラブルシューティング

このページには、一般的な問題に対する原因メッセージの例と詳細情報へのリンクが一覧表示されます。いくつかのチェックにわたって持続的に状態に問題があることが検出されると、原因メッセージが Elastic Beanstalk コンソールの環境の概要ページに表示され、イベントに記録されます。

デプロイ

Elastic Beanstalk は、環境のデプロイ後の整合性を監視します。ローリングデプロイに失敗した場合、環境のインスタンスで実行されているアプリケーションバージョンが異なっている可能性があります。これは、デプロイが 1 つあるいは複数のバッチで成功しても、すべてのバッチへのデプロイが完了する前に失敗した場合に起こります。

5 つのインスタンスのうち 2 つで正しくないアプリケーションバージョンが検出された。予想されるバージョン「v1」(デプロイ 1)。

環境インスタンスのアプリケーションバージョンが正しくない。予想されるバージョン「v1」(デプロイ 1)。

予想されるアプリケーションバージョンが、環境内のいくつかの、あるいはすべてのインスタンスで起動していない。

正しくないアプリケーションバージョン「v2」(デプロイ 2)。予想されるバージョン「v1」(デプロイ 1)。

インスタンスにデプロイされているアプリケーションが、予想されるバージョンとは異なります。デプロイが失敗すると、予想されるバージョンがもっとも直近の成功したデプロイのバージョンにリセットされます。上記の例では、最初のデプロイ (バージョン「v1」) は成功し、2 番目デプロイ (バージョン「v2」) は失敗しています。「v2」を実行するインスタンスは、正常な状態ではないみなされます。

この問題を解決し、別のデプロイを開始します。機能していた前のバージョンを再度デプロイするか、環境をデプロイ中はヘルスチェックを無視するように設定し、デプロイを強制的に完了させるため新しいバージョンを再度デプロイします。

また、間違ったアプリケーションバージョンを実行しているインスタンスを特定し、終了することもできます。Elastic Beanstalk は、正しいバージョンのインスタンスを起動し、終了させたインスタンスと置き換えます。EB CLI ヘルスコマンドを使って、間違ったアプリケーションバージョンを実行しているインスタンスを識別します。

アプリケーションサーバー

15% of requests are erroring with HTTP 4xx

20% of the requests to the ELB are erroring with HTTP 4xx.

インスタンスまたは環境に対する HTTP リクエストの多くが 4xx エラーを原因として失敗しています。

400 シリーズのステータスコードは、ユーザーが存在しないページをリクエスト (404 File Not Found) するなどの不適切なリクエストをした、またはユーザーにはアクセスする権利がない (403 Forbidden) ことを示します。404 の数が低いことは珍しくありませんが、その数が多いと、存在しないページへの内部リンクまたは外部リンクがあることを意味する場合があります。このような問題は、不適切な内部リンクを修正し、不適切な外部リンクにリダイレクトを追加することによって解決できます。

5% of the requests are failing with HTTP 5xx

3% of the requests to the ELB are failing with HTTP 5xx.

インスタンスまたは環境に対する HTTP リクエストの多くが 500 シリーズのステータスコードを原因として失敗しています。

500 シリーズのステータスコードは、アプリケーションサーバーで内部エラーが発生したことを示します。このような問題は、アプリケーションコードにエラーがあり、迅速にエラーを特定し、修正する必要があることを示します。

95% of CPU is in use

インスタンスについて、ヘルスエージェントが非常に高い CPU 使用率をレポートしており、インスタンスのヘルスステータスが Warning または Degraded に設定されています。

環境をスケールしてインスタンスの負荷を軽減します。

ワーカーインスタンス

20 messages waiting in the queue (25 seconds ago)

リクエストの処理速度よりも速くリクエストがワーカー環境のキューに追加されています。環境をスケールして処理能力を向上させます。

5 messages in Dead Letter Queue (15 seconds ago)

ワーカーリクエストが繰り返し失敗しており、the section called “デッドレターキュー” に追加されています。デッドレターキュー内のリクエストをチェックして、失敗している理由を確認します。

その他のリソース

4 active instances is below Auto Scaling group minimum size 5

環境内で実行されているインスタンスの数が、Auto Scaling グループに対して設定されている最小数に達していません。

Auto Scaling group (groupname) notifications have been deleted or modified

Auto Scaling グループに対して設定されている通知が Elastic Beanstalk 外部で修正されています。

アラームの管理

このトピックでは、モニタリングしているメトリクスのアラームを作成する手順について説明します。また、既存のアラームを表示し、その状態を確認する手順も示します。

Elastic Beanstalk コンソールを使用して、モニタリングしているメトリクスに対するアラームを作成できます。アラームは、 AWS Elastic Beanstalk 環境の変化をモニタリングするのに役立ちます。これにより、問題が発生する前に問題を簡単に特定して軽減できます。たとえば、環境の CPU 使用率が特定のしきい値を超えた場合に通知するアラームを設定し、潜在的な問題が発生する前に気づくことができます。詳細については、「「Amazon CloudWatch で Elastic Beanstalk を使用する」」を参照してください。

> **Note**
>
> Elastic Beanstalk はモニタリングとアラームに CloudWatch を使用します。つまり、CloudWatch のコストは、使用するアラームの AWS アカウントに適用されます。

特定のメトリクスのモニタリングの詳細については、「ベーシックヘルスレポート」を参照してください。

アラームの状態をチェックする

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[Alarms] (アラーム) を選択します。

   このページには、既存のアラームのリストが表示されます。
   アラーム状態にあるアラームには、警告アイコン
   ( )
   のフラグが立ちます。
4. アラームをフィルタリングするには、ドロップダウンメニューを選択してから、[フィルター] を選択します。
5. アラームを編集または削除するには、それぞれ編集アイコン
   ( )
   または削除アイコン
   ( )
   を選択します。

アラームを作成する

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[モニタリング] を選択します。
4. アラームを作成するメトリクスを見つけて、アラームアイコン
   (　　)
   を選択します。[アラームの追加] ページが表示されます。
5. アラームについての詳細を入力します:
   - [Name]: このアラームの名前。
   - [Description] (オプション): このアラームの内容の短い説明。
   - [Period]: 読み取り間隔。
   - [Threshold]: アラームがトリガーされるためのメトリクスの動作と超えるべき値を指定します。
   - [Change state after]: アラームの状態変更をトリガーするしきい値を超えた後の時間。
   - [通知]: アラームの状態が変更された時に通知される Amazon SNS トピック。
   - [Notify when state changes to]:
     - [OK]: メトリクスの値が、定義されたしきい値の範囲内にあります。
     - [Alarm]: 定義したしきい値を超えたメトリクス。
     - [Insufficient data]: アラームが開始されたか、メトリクスが利用可能でないか、またはメトリクスがアラームの状態を決定するためのデータが不足しています。
6. [Add] (追加) を選択します。環境の更新中は環境ステータスがグレーになります。作成したアラームを表示するには、ナビゲーションペインで [Alarms (アラーム)] を選択します。

Elastic Beanstalk 環境の変更履歴の表示

このトピックでは、Elastic Beanstalk 環境に対して行われた設定変更の履歴を Elastic Beanstalk コンソールを使用して表示する方法について説明します。

Elastic Beanstalk は、AWS CloudTrail に記録されたイベントから変更履歴を抽出し、それらをナビゲーションとフィルタリングが容易な形で一覧表示します。

[変更履歴] パネルには、環境に対し加えられた変更に関する、次のような情報が表示されます。

- 変更が行われた日付と時刻
- 変更を担当した IAM ユーザー

- 変更に使用されたソースツール (Elastic Beanstalk コマンドラインインターフェイス (EB CLI) またはコンソール)
- 設定パラメータと、それに対し設定された新しい値

データベースユーザーの名前など、変更が行われたことにより影響を受ける機密データは、変更の内容に含まれていてもパネルには表示されません。

変更履歴を表示するには

1. Elastic Beanstalk 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 [変更履歴] をクリックします。

   [変更履歴] ページには、Elastic Beanstalk 環境に対して行われた設定変更のリストが表示されます。

このページの情報の操作に関する以下の点に注意してください。

- [<] (前へ) または [>] (次へ) をクリックするか、特定のページ番号を選択して、リスト内のページを切り替えて表示することができます。
- [設定変更] 列で矢印アイコンをクリックすると、[行われた変更] 見出しの下の変更内容のリストを、展開したり折りたたんだりできます。
- 検索バーを使用すると、変更履歴リストから結果をフィルタリングできます。任意の文字列を入力して、一覧表示される変更の内容を絞り込むことが可能です。

表示結果のフィルタリングについては、次の点に注意してください。

- 検索のフィルタリングでは、大文字と小文字が区別されません。
- [設定変更] 列の情報に基づいて、表示される変更内容をフィルタリングできます。これは、[行われた変更] 内で折りたたまれているために表示されていない場合でも適用されます。
- フィルタできるのは、そこに表示されている結果のみです。ただし、さらに多くの結果を表示するために別のページに移動した場合でも、フィルターは同じ状態に維持されます。フィルタリングされた結果は、移動先のページの結果セットにも追加されます。

以下の例は、以前の画面に表示されたデータをフィルタする方法を示しています。

- 検索ボックスに **`GettingStartedApp-env`** と入力すると、GettingStartedApp-env という名前の環境に加えられた変更のみを表示するように結果が絞り込まれます。
- 検索ボックスに **`example3`** と入力すると、ユーザー名に example3 という文字列を含む IAM ユーザーによって行われた変更のみを表示するように結果が絞り込まれます。
- 検索ボックスに **`2020-10`** と入力した場合は、2020 年 10 月に行われた変更のみを表示するように結果が絞り込まれます。結果をさらに絞り込み、2020 年 10 月 16 日に行われた変更のみを表示するには、検索ボックスの値を **`2020-10-16`** に変更します。
- また、aws:elasticbeanstalk:environment:proxy:staticfiles という名前のネームスペースに加えられた変更のみが表示されるように結果を絞り込むには、検索ボックスに **`proxy:staticfiles`** と入力します。フィルタリングされた結果が、行として表示されます。これは、[行われた変更] に折りたたまれている結果に対しても当てはまります。

Elastic Beanstalk 環境のイベントストリームの表示

このトピックでは、アプリケーションに関連付けられたイベントや通知にアクセスする方法について説明します。

Elastic Beanstalk コンソールでのイベントの表示

Elastic Beanstalk コンソールでイベントを表示するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインの [Events] を選択してください。

   イベントページに、環境で記録されたすべてのイベントが一覧表示されます。[<] (前)、[>] (次)、または [ページ番号] を選択して、一覧からページを移動できます。[重要度] ドロップダウンリストを使用すると、イベントのタイプをフィルターできます。

コマンドラインツールを使用してイベントを表示する

EB CLI と AWS CLI はどちらも、イベントを取得するためのコマンドを提供しています。EB CLI を使用して環境を管理する場合に、イベントの一覧を出力するには、eb events を使用します。このコマンドには、Ctrl+C を押して出力を停止するまで新しいイベントの表示を続行する `--follow` オプションもあります。

を使用してイベントをプルするには AWS CLI、 `describe-events` コマンドを使用して、名前または ID で環境を指定します。

```
$ aws elasticbeanstalk describe-events --environment-id e-gbjzqccra3
{
    "Events": [
        {
            "ApplicationName": "elastic-beanstalk-example",
            "EnvironmentName": "elasticBeanstalkExa-env",
            "Severity": "INFO",
            "RequestId": "a4c7bfd6-2043-11e5-91e2-9114455c358a",
            "Message": "Environment update completed successfully.",
            "EventDate": "2015-07-01T22:52:12.639Z"
        },
...
```

コマンドラインツールの詳細については、「???」を参照してください。

サーバーインスタンスの一覧表示と接続

このトピックでは、Elastic Beanstalk アプリケーション環境を実行している Amazon EC2 インスタンスのリストを表示する方法と、それらに接続する方法について説明します。

Elastic Beanstalk コンソールを使用して、 AWS Elastic Beanstalk アプリケーション環境を実行している Amazon EC2 インスタンスのリストを表示できます。インスタンスは、SSH クライアントを使用して接続できます。Windows の Remote Desktop を使用してインスタンスに接続することもできます。

> **Important**
>
> Elastic Beanstalk によってプロビジョニングされた Amazon EC2 インスタンスにアクセスする前に、Amazon EC2 キーペアを作成し、Elastic Beanstalk によってプロビジョニングされた Amazon EC2インスタンスを、Amazon EC2 キーペアを使用するように設定する必要があります。Amazon EC2 キーペアをセットアップするには、AWS マネジメントコンソールを使用します。Amazon EC2 のキーペアを作成する方法については、Amazon EC2 入門ガイドを参照してください。Amazon EC2 インスタンスを設定して Amazon EC2 キーペアを使用する方法の詳細については、EC2 キーペア を参照してください。
> デフォルトでは、Elastic Beanstalk は、レガシー Windows コンテナ以外の Windows コンテナ内にある EC2 インスタンスへのリモート接続を有効にしません。(Elastic Beanstalk

は、RDP 接続用にポート 3389 を使用するようにレガシー Windows コンテナ内の EC2 インスタンスを設定します)。インスタンスへのインバウンドトラフィックを許可するセキュリティグループにルールを追加することにより、Windows を実行している EC2 インスタンスへのリモート接続の有効にすることができます。リモート接続を終了するときにこのルールを削除することを強くお勧めします。次回リモートでログインする必要があるときには、ルールを再度追加できます。詳細については、『Microsoft Windows 用 Amazon Elastic Compute Cloud ユーザーガイド』の「Windows インスタンスに対するインバウンド RDP トラフィックのルールの追加」および「Windows インスタンスへの接続」を参照してください。

環境の Amazon EC2 インスタンスを表示して接続するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. コンソールのナビゲーションペインで、[Load Balancers] を選択します。
3. Elastic Beanstalk によって作成されたロードバランサーの名前には [awseb] が含まれます。お使いの環境のロードバランサーを探し、クリックしてください。
4. コンソールの下部のペインで、[Instances] タブを選択します。

   Elastic Beanstalk 環境のロードバランサーが使用するインスタンスのリストが表示されます。接続したいインスタンス ID をメモしてください。
5. Amazon EC2 コンソールのナビゲーションペインで [Instances (インスタンス)] を選択し、リストで目的のインスタンス ID を見つけます。
6. お使いの環境のロードバランサーで実行されている Amazon EC2 インスタンスのインスタンス ID を右クリックし、[Connect (接続)] を選択します。
7. [Description] タブにある、インスタンスのパブリック DNS アドレスをメモしてください。
8. 選択した SSH クライアントを使用して Linux を実行しているインスタンスに接続するには、[ssh -i .ec2/mykeypair.pem ec2-user@<public-DNS-of-the-instance>] と入力します。

Amazon EC2 Linux インスタンスへの接続の詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 Linux インスタンスの開始方法」を参照してください。

Elastic Beanstalk 環境で .NET on Windows Server プラットフォームを使用している場合は、Amazon EC2 ユーザーガイド」の「Amazon EC2 Windows インスタンスの開始方法」を参照してください。 Amazon EC2

Elastic Beanstalk 環境の Amazon EC2 インスタンスからのログの表示

このトピックでは、Elastic Beanstalk が提供するインスタンスログのタイプについて説明します。また、これらの取得と管理に関する詳細な手順についても説明します。

Elastic Beanstalk 環境の Amazon EC2 インスタンスでは、アプリケーションまたは設定ファイルに関する問題を解決する際に確認するためのログが生成されます。ウェブサーバー、アプリケーションサーバー、Elastic Beanstalk プラットフォームスクリプトによって作成されたログ AWS CloudFormation は、個々のインスタンスにローカルに保存されます。それらは、環境管理コンソールまたは EB CLI を使用して簡単に検索できます。また、Amazon CloudWatch Logs にリアルタイムでログがストリーミングされるように環境を設定することもできます。

テールログは、最も一般的に使用されるログファイル (Elastic Beanstalk 運用ログ、ウェブサーバーやアプリケーションサーバーのログ) の最後の 100 行です。環境マネジメントコンソールまたは eb logs を使用してテールログをリクエストすると、環境のインスタンスにより、最新のログエントリを単一のテキストファイルに連結したものが Amazon S3 にアップロードされます。

バンドルログは、yum および cron のログおよび AWS CloudFormationの複数のログを含むさまざまなログファイルのフルログです。バンドルログをリクエストすると、環境内のインスタンスでは完全ログファイルが ZIP アーカイブとしてパッケージ化され、Amazon S3 にアップロードされます。

ローテーションされたログを Amazon S3 にアップロードするには、環境内のインスタンスの インスタンスプロファイルに、Elastic Beanstalk Amazon S3 バケットに書き込むためのアクセス許可が必要になります。これらのアクセス許可は、デフォルトのインスタンスプロファイルに含まれています。このプロファイルは、Elastic Beanstalk コンソールで初めて環境を起動する際に、作成するよう Elastic Beanstalk から求められるものです。

インスタンスログを取得するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで [ログ] を選択します。
4. [ログのリクエスト] を選択し、取得するログの種類を選択します。ログ末尾を取得するには、[Last 100 Lines] を選択します。バンドルログを取得するには、[Full Logs] を選択します。
5. Elastic Beanstalk でログの取得が完了したら、[ダウンロード] を選択します。

Elastic Beanstalk は、テールログとバンドルログを Amazon S3 バケットに保存し、ログにアクセスするために使用できる署名済みの Amazon S3 URL を生成します。Elastic Beanstalk は、15 分の持続時間が経過すると、Amazon S3 からファイルを削除します。

> **Warning**
>
> このファイルには、署名済み Amazon S3 URL を所有していれば誰でも、削除前にアクセスできます。URL を使用できるように指定するのは、信頼されたパーティに対してのみにしてください。

> **Note**
>
> ユーザーポリシーには、`s3:DeleteObject` アクセス許可が必要です。Elastic Beanstalk は、ユーザーのアクセス許可を使用して Amazon S3 からログを削除します。

ログを保持するためには、ログのローテーション後、自動的に Amazon S3 に発行されるように環境を設定できます。Amazon S3 へのログのローテーションを有効にするには、「インスタンスログ表示の設定」の手順に従ってください。環境のインスタンスは 1 時間に一度ローテーションされるログをアップロードしようと試みます。

アプリケーションが、環境のプラットフォームのデフォルトの設定の一部ではない場所にあるログを生成する場合、設定ファイル (`.ebextensions`) を使用してデフォルトの設定を拡張できます。アプリケーションのログファイルをログ末尾、バンドルログ、またはログローテーションに追加できます。

ログのリアルタイムストリーミングや長期保存を行う場合は、Amazon CloudWatch Logs にログがストリーミングされるよう環境を設定します。

セクション

- Amazon EC2 インスタンス上のログの場所
- Amazon S3 のログの場所
- Linux でのログのローテーション設定
- デフォルトのログタスク設定の拡張
- Amazon CloudWatch Logs へのログファイルのストリーミング

Amazon EC2 インスタンス上のログの場所

ログは、環境内の Amazon EC2 インスタンスで標準の場所に保存されます。Elastic Beanstalk では、以下のログが生成されます。

Amazon Linux 2

- `/var/log/eb-engine.log`

Amazon Linux AMI (AL1)

> **Note**
>
> 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

- `/var/log/eb-activity.log`
- `/var/log/eb-commandprocessor.log`

Windows Server

- `C:\Program Files\Amazon\ElasticBeanstalk\logs\`
- `C:\cfn\log\cfn-init.log`

これらのログには、設定ファイルに関するメッセージなど、デプロイメントアクティビティに関するメッセージが含まれます (`.ebextensions`)。

各アプリケーションとウェブサーバーは、固有フォルダにログを保存します。

- Apache – `/var/log/httpd/`
- IIS – `C:\inetpub\wwwroot\`
- Node.js – `/var/log/nodejs/`
- nginx – `/var/log/nginx/`

- Passenger – `/var/app/support/logs/`
- Puma – `/var/log/puma/`
- Python – `/opt/python/log/`
- Tomcat – `/var/log/tomcat/`

Amazon S3 のログの場所

環境のテールログまたはバンドルログをリクエストした場合や、ローテーションされたログをインスタンスがアップロードした場合、ログは Amazon S3 の Elastic Beanstalk バケットに格納されます。Elastic Beanstalk は、環境を作成する AWS リージョン`elasticbeanstalk-`*`region`*`-`*`account-id`*ごとに という名前のバケットを作成します。このバケット内では、ログはパス `resources/environments/logs/`*`logtype`*`/`*`environment-id`*`/`*`instance-id` 内に保存されます。

たとえば、`us-west-2`アカウント の AWS リージョン `i-0a1fd158`の Elastic Beanstalk 環境`e-mpcwnwheky`のインスタンス からのログは`123456789012`、次の場所に保存されます。

- テールログ –

  `s3://elasticbeanstalk-us-west-2-123456789012/resources/environments/logs/tail/e-mpcwnwheky/i-0a1fd158`
- バンドルログ –

  `s3://elasticbeanstalk-us-west-2-123456789012/resources/environments/logs/bundle/e-mpcwnwheky/i-0a1fd158`
- ローテーションされたログ –

  `s3://elasticbeanstalk-us-west-2-123456789012/resources/environments/logs/publish/e-mpcwnwheky/i-0a1fd158`

> **Note**
>
> 環境 ID は、環境管理コンソールに表示されます。

Elastic Beanstalk は、テールログとバンドルログを作成から 15 分後に Amazon S3 から自動的に削除します。ローテーションされたログは、削除するか S3 Glacier に移動するまで保持されます。

Linux でのログのローテーション設定

Linux プラットフォームでは、Elastic Beanstalk は `logrotate` を使用して定期的にログのローテーションを行います。設定されている場合、ログがローカルでローテーションされると、そのログはローテーションタスクにより Amazon S3 にアップロードされます。ローカルにローテーションされたログは、ログの末尾またはバンドルログにはデフォルトで表示されません。

`logrotate` 用の Elastic Beanstalk 設定ファイルは `/etc/logrotate.elasticbeanstalk.hourly/` にあります。これらのローテーション設定はプラットフォームに固有で、今後のプラットフォームのバージョンで変更される場合があります。使用できる設定の設定と例の詳細については、`man logrotate` を実行してください。

設定ファイルは、`/etc/cron.hourly/` の cron ジョブで呼び出されます。`cron` の詳細については、「`man cron`」を実行してください。

デフォルトのログタスク設定の拡張

Elastic Beanstalk は Amazon EC2 インスタンス上の `/opt/elasticbeanstalk/tasks` (Linux) または `C:\Program Files\Amazon\ElasticBeanstalk\config` (Windows Server) のサブフォルダにあるファイルを使用して、テールログ、バンドルログ、およびログローテーションを設定します。

Amazon Linux で

- テールログ –

  `/opt/elasticbeanstalk/tasks/taillogs.d/`
- バンドルログ –

  `/opt/elasticbeanstalk/tasks/bundlelogs.d/`
- ローテーションされたログ –

  `/opt/elasticbeanstalk/tasks/publishlogs.d/`

Windows Server の場合:

- テールログ –

  `c:\Program Files\Amazon\ElasticBeanstalk\config\taillogs.d\`

- バンドルログ –

  `c:\Program Files\Amazon\ElasticBeanstalk\config\bundlelogs.d\`
- ローテーションされたログ –

  `c:\Program Files\Amazon\ElasticBeanstalk\config\publogs.d\`

たとえば、Linux のファイル `eb-activity.conf` が 2 つのログファイルを末尾ログのタスクに追加します。

/opt/elasticbeanstalk/tasks/taillogs.d/eb-activity.conf

```
/var/log/eb-commandprocessor.log
/var/log/eb-activity.log
```

環境設定ファイル (`.ebextensions`) を使用して、独自の `.conf` ファイルをこれらのフォルダに追加できます。`.conf` ファイルでは、お客様のアプリケーションに固有のログファイルが表示されます。これは Elastic Beanstalk によってログファイルタスクに追加されます。

`files` セクションを使用して、変更するタスクに設定ファイルを追加します。たとえば、次の設定テキストは、ログ設定ファイルを環境の各インスタンスに追加します。このログ設定ファイル `cloud-init.conf` は、ログ末尾に `/var/log/cloud-init.log` を追加します。

```
files:
  "/opt/elasticbeanstalk/tasks/taillogs.d/cloud-init.conf" :
    mode: "000755"
    owner: root
    group: root
    content: |
      /var/log/cloud-init.log
```

このテキストを `.config` ファイル名拡張子をもつファイルに追加して `.ebextensions` という名前のフォルダ内のソースバンドルに追加します。

```
~/workspace/my-app
|-- .ebextensions
|   `-- tail-logs.config
|-- index.php
`-- styles.css
```

Linux プラットフォームでは、ログのタスク設定でワイルドカード文字を使用することもできます。この設定ファイルは `.log` ファイル名拡張子をもつすべてのファイルをアプリケーションのルートにある `log` フォルダからバンドルログに追加します。

```
files:
  "/opt/elasticbeanstalk/tasks/bundlelogs.d/applogs.conf" :
    mode: "000755"
    owner: root
    group: root
    content: |
      /var/app/current/log/*.log
```

ログタスク設定では、Windows プラットフォームでワイルドカード文字はサポートされません。

> **Note**
>
> ログのカスタマイズ手順に慣れるために、EB CLI を使用してサンプルアプリケーションをデプロイできます。このために、EB CLI によって、サンプル設定の `.ebextentions` サブディレクトリを含むローカルアプリケーションディレクトリが作成されます。サンプルアプリケーションのログファイルを使用して、このトピックで説明されているログ取得機能をさまざまに試すこともできます。

設定ファイルの使用方法の詳細については、「設定ファイル (`.ebextensions`) による高度な環境のカスタマイズ」を参照してください。

ログ末尾やバンドルログを拡張するのと同様に、設定ファイルを使用してログのローテーションを拡張できます。Elastic Beanstalk で、独自のログのローテーションと Amazon S3 へのアップロードが行われる際には、追加のログもローテーションおよびアップロードされます。ログのローテーション拡張は、プラットフォームのオペレーティングシステムによって動作が異なります。以下のセクションでは、2 つのケースについて説明します。

Linux でのログのローテーション拡張

「Linux でのログのローテーション設定」で説明したように、Linux プラットフォームでは Elastic Beanstalk が `logrotate` を使用してログをローテーションします。ログのローテーションについてアプリケーションのログファイルを設定すると、アプリケーションはログファイルのコピーを作成する必要はありません。Elastic Beanstalk は `logrotate` を設定し、ローテーションごとにアプリケーションのログファイルのコピーを作成します。したがって、アプリケーションはログファイルにアクティブに書き込んでいないときは、ログファイルをロック解除した状態を維持する必要があります。

Windows サーバーでのログのローテーション拡張

Windows Server では、アプリケーションのログファイルでログのローテーションを設定する場合、アプリケーションはログファイルを定期的にローテーションする必要があります。Elastic Beanstalk は、設定されたパターンで始まる名前を持つファイルを探し、Amazon S3 へのアップロードのためにそれらのファイルを選択します。さらに、ファイル名のピリオドは無視され、Elastic Beanstalk は、名前のピリオドまでがログファイルのベース名であると見なします。

Elastic Beanstalk は、最新のものを除くすべてのバージョンのベースログファイルをアップロードします。これは、最新のファイルがアクティブなアプリケーションログファイルであり、ロックされる可能性があると見なされるためです。したがって、アプリケーションはローテーション間でアクティブなログファイルのロックを維持できます。

たとえば、アプリケーションが `my_log.log` というログファイルに書き込み、この名前を `.conf` ファイルで指定するとします。アプリケーションは定期的にファイルをローテーションします。アプリケーションは Elastic Beanstalk のローテーションサイクル中に、ログファイルのフォルダーで `my_log.log`、`my_log.0800.log`、`my_log.0830.log` の各ファイルを検索します。Elastic Beanstalk は、すべてがベース名 `my_log` のバージョンであると見なします。ファイル `my_log.log` の変更時刻が最新であるため、Elastic Beanstalk は他の 2 つのファイル `my_log.0800.log` および `my_log.0830.log` のみをアップロードします。

Amazon CloudWatch Logs へのログファイルのストリーミング

Amazon CloudWatch Logs にログがストリーミングされるように環境を設定するには、Elastic Beanstalk コンソールまたは設定オプションを使用します。CloudWatch Logs では、環境の各インスタンスがロググループにログをストリーミングします。ロググループは、環境の終了後も数週間または数年間保持するよう設定できます。

ストリーミングされるログのセットは環境によって異なりますが、アプリケーションの前面で実行される nginx または Apache プロキシサーバーからの `eb-engine.log` とアクセスログが常に含まれます。

Elastic Beanstalk コンソールでは、環境の作成時または既存の環境のいずれかに、ログのストリーミングを設定できます。コンソールから次のオプションを設定できます。CloudWatch Logs へのログストリーミングの有効化/無効化、保持日数の設定、Lifecyle オプションからの選択。次の例では、環境が終了した場合でもログは最大 7 日間保存されます。

次の設定ファイルでは、環境が終了した場合でも 180 日間はログストリーミングが可能になります。

Example .ebextensions/log-streaming.config

```
option_settings:
  aws:elasticbeanstalk:cloudwatch:logs:
    StreamLogs: true
    DeleteOnTerminate: false
    RetentionInDays: 180
```

他の AWS サービスでの Elastic Beanstalk の使用

このセクションのトピックでは、Elastic Beanstalk アプリケーションで追加の AWS サービスを使用するさまざまな方法について説明します。アプリケーションの環境を実装するために、Elastic Beanstalk は他の AWS サービスのリソースを管理するか、その機能を使用します。さらに、Elastic Beanstalk は、環境の一部として直接使用しない AWS サービスと統合されます。

トピック

- アーキテクチャの概要
- Amazon CloudFront で Elastic Beanstalk を使用する
- を使用した Elastic Beanstalk API コールのログ記録 AWS CloudTrail
- 「Amazon CloudWatch で Elastic Beanstalk を使用する」
- Amazon CloudWatch Logs で Elastic Beanstalk を使用する
- Amazon EventBridge で Elastic Beanstalk を使用する
- を使用した Elastic Beanstalk リソースの検索と追跡 AWS Config
- Amazon DynamoDB で Elastic Beanstalk を使用する
- Amazon ElastiCache で Elastic Beanstalk を使用するには
- Amazon Elastic File System で Elastic Beanstalk を使用する
- での Elastic Beanstalk の使用 AWS Identity and Access Management
- Amazon RDS で Elastic Beanstalk を使用する
- Amazon S3 で Elastic Beanstalk を使用する
- AWS Secrets Manager および Parameter Store での Elastic Beanstalk AWS Systems Manager の使用
- Amazon VPC で Elastic Beanstalk を使用する

アーキテクチャの概要

次の図は、Amazon CloudFront、Amazon Simple Storage Service (Amazon S3)、Amazon Relational Database Service (Amazon RDS) などの他の AWS 製品と連携する複数のアベイラビリティーゾーンにわたる Elastic Beanstalk のアーキテクチャの例を示しています。 Amazon Relational Database Service

耐障害性に備えて、N+1 Amazon EC2 インスタンスを確保し、複数のアベイラビリティーゾーンにインスタンスを分散させることをお勧めします。アベイラビリティーゾーンが故障することはあまりありませんが、たとえ故障しても、別のアベイラビリティーゾーンで実行されている Amazon EC2 インスタンスを引き続き利用できます。最小インスタンス数と複数のアベイラビリティーゾーンに合わせて、Amazon EC2 Auto Scaling を調整することができます。これを行う手順については、「Elastic Beanstalk 環境インスタンスの Auto Scaling 」を参照してください。耐障害性を備えたアプリケーション構築の詳細については、「AWSで耐障害性の高いアプリケーションの構築」を参照してください。

次のセクションでは、Amazon CloudFront、Amazon CloudWatch、Amazon DynamoDB Amazon ElastiCache、Amazon RDS、Amazon Route 53、Amazon Simple Storage Service、Amazon VPC、IAM との統合について詳しく説明します。

Amazon CloudFront で Elastic Beanstalk を使用する

Amazon CloudFront は、静的および動的なウェブコンテンツ（.html、.css、.php、イメージ、メディアファイルなど）をエンドユーザーに高速に配信するウェブサービスです。CloudFront は、エッジロケーションの世界的ネットワークを経由してコンテンツを配信します。CloudFront を使用して提供されているコンテンツをエンドユーザーが要求すると、そのユーザーはエッジロケーションにルーティングされます。エッジロケーションでは、最も低いレイテンシーが提供されるので、コンテンツは、可能な最高のパフォーマンスで配信されます。コンテンツがこのエッジロケーションにすでに存在する場合、CloudFront はコンテンツを直ちに配信します。コンテンツがこのエッジロケーションに現在存在しない場合、CloudFront は、コンテンツの最終バージョンのソースとして識別されている Amazon S3 バケットまたは HTTP サーバー（ウェブサーバーなど）からコンテンツを取り込みます。

Elastic Beanstalk アプリケーションを作成してデプロイしたら、CloudFront にサインアップし、CloudFront を使用してコンテンツを配信することができます。CloudFront の詳細については、Amazon CloudFront 開発者ガイドを参照してください。

を使用した Elastic Beanstalk API コールのログ記録 AWS CloudTrail

Elastic Beanstalk は AWS CloudTrail、Elastic Beanstalk のユーザー、ロール、または 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、Elastic Beanstalk コンソールから、EB CLI から、および Elastic Beanstalk へのコードからの呼び出しを含む、Elastic Beanstalk 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。証跡

を作成する場合は、Elastic Beanstalk のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的デリバリー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、Elastic Beanstalk 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 履歴の Elastic Beanstalk 情報

CloudTrail は、 AWS アカウントの作成時にアカウントで有効になります。Elastic Beanstalk 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。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。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。

Elastic Beanstalk のイベントなど、 AWS アカウント内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、ログファイルを CloudTrail で Amazon S3 バケットに配信できます。デフォルトでは、コンソールで追跡を作成するときに、追跡がすべてのリージョンに適用されます。証跡は AWS パーティション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より詳細に分析し、それに基づいて行動するように、他の AWS サービスを設定できます。詳細については、以下を参照してください。

- 証跡を作成するための概要
- 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 Elastic Beanstalk アクションは CloudTrail が記録します。これらのアクションは、AWS Elastic Beanstalk API リファレンスで説明されています。例えば、`DescribeApplications`、`UpdateEnvironment`、`ListTagsForResource` 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。

Elastic Beanstalk ログファイルエントリの概要

「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 のログファイルは、単一か複数のログエントリを含みます。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、リクエストのパラメータなどの情報が含まれます。CloudTrail ログファイルは、公開 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。

次の例は、`sample-app` アプリケーションの `sample-env` 環境について、`intern` という名前の IAM ユーザーが呼び出した `UpdateEnvironment` アクションを示す CloudTrail ログエントリを示しています。

```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "AIXDAYQEXAMPLEUMLYNGL",
      "arn": "arn:aws:iam::123456789012:user/intern",
      "accountId": "123456789012",
      "accessKeyId": "ASXIAGXEXAMPLEQULKNXV",
      "userName": "intern",
      "sessionContext": {
        "attributes": {
          "mfaAuthenticated": "false",
          "creationDate": "2016-04-22T00:23:24Z"
        }
      },
      "invokedBy": "signin.amazonaws.com"
    },
    "eventTime": "2016-04-22T00:24:14Z",
    "eventSource": "elasticbeanstalk.amazonaws.com",
    "eventName": "UpdateEnvironment",
```

```
    "awsRegion": "us-west-2",
    "sourceIPAddress": "255.255.255.54",
    "userAgent": "signin.amazonaws.com",
    "requestParameters": {
      "applicationName": "sample-app",
      "environmentName": "sample-env",
      "optionSettings": []
    },
    "responseElements": null,
    "requestID": "84ae9ecf-0280-17ce-8612-705c7b132321",
    "eventID": "e48b6a08-c6be-4a22-99e1-c53139cbfb18",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  }]
}
```

「Amazon CloudWatch で Elastic Beanstalk を使用する」

Amazon CloudWatch では、さまざまなメトリクスをモニタリング、管理、および発行し、メトリクスのデータに基づいてアラームアクションを設定できます。Amazon CloudWatch モニタリングにより、システムとアプリケーションのメトリクスを収集、分析、および表示できます。これにより、自信を持って本番上およびビジネス上の決定を下すことができます。

Amazon CloudWatch を使用すると、Amazon EC2 インスタンスのパフォーマンスなどのアマゾン ウェブ サービス (AWS) リソースのメトリクスを収集できます。独自のメトリクスを Amazon CloudWatch に対して直接発行することもできます。Amazon CloudWatch アラームは、定義したルールに基づいて通知を送信したり、モニタリング対象のリソースを自動的に変更したりするための機能です。これにより、決定事項がさらに実装しやすくなります。例えば、ご自身の代わりに Amazon EC2 Auto Scaling および Amazon Simple Notification Service (Amazon SNS) アクションを開始するアラームを作成できます。

Elastic Beanstalk は Amazon CloudWatch を自動的に使用して、アプリケーションや環境のステータスのモニタリングに役立てます。Amazon CloudWatch コンソールに移動し、ダッシュボードを表示すると、すべてのリソースとアラームの概要情報を入手できます。また、メトリクスの詳細を表示して、カスタムメトリクスを追加することもできます。

Amazon CloudWatch の詳細については、Amazon CloudWatch 開発者ガイドを参照してください。Amazon CloudWatch を Elastic Beanstalk で使用する方法の例については、「the section called “例: カスタム Amazon CloudWatch メトリクスの使用”」を参照してください。

Amazon CloudWatch Logs で Elastic Beanstalk を使用する

このトピックでは、Amazon CloudWatch Logs サービスが Elastic Beanstalk に提供できるモニタリング機能について説明します。また、設定について順を追って説明し、各 Elastic Beanstalk プラットフォームのログの場所を一覧表示します。

CloudWatch Logs を実装すると、次のモニタリングアクティビティを実行できます。

- 環境の Amazon EC2 インスタンスから Elastic Beanstalk アプリケーション、システム、およびカスタムのログファイルをモニタリングおよびアーカイブできます。
- アラームを設定すると、メトリクスフィルタが抽出する特定のログストリームイベントに簡単に反応できるようになります。

環境内の各 Amazon EC2 インスタンスにインストールされた CloudWatch Logs エージェントは、設定した各ロググループの CloudWatch サービスに対してメトリクスのデータポイントを発行します。各ロググループは、独自のフィルターパターンを適用して、どのログストリームイベントをデータポイントとして CloudWatch に送信するかを決定します。同じロググループに属するログストリームは、保持、監視、アクセス制御について同じ設定を共有します。「CloudWatch Logs へのインスタンスログのストリーミング」に示すように、CloudWatch サービスに自動的にログをストリーミングするよう Elastic Beanstalk を設定できます。用語や概念を含む CloudWatch Logs の詳細については、『Amazon CloudWatch Logs ユーザーガイド』を参照してください。

インスタンスログに加えて、環境の拡張ヘルスを有効にすると、CloudWatch Logs にヘルス情報をストリーミングするように環境を設定できます。「Amazon CloudWatch Logs への Elastic Beanstalk 環境ヘルス情報のストリーミング」を参照してください。

トピック

- CloudWatch Logs へのログストリーミングの前提条件
- Elastic Beanstalk が CloudWatch Logs を設定する方法
- CloudWatch Logs へのインスタンスログのストリーミング
- CloudWatch Logs 統合のトラブルシューティング
- Amazon CloudWatch Logs への Elastic Beanstalk 環境ヘルス情報のストリーミング

CloudWatch Logs へのログストリーミングの前提条件

環境の Amazon EC2 インスタンスから CloudWatch Logs へのログのストリーミングを有効にするには、以下の条件を満たす必要があります。

- プラットフォーム – この機能はこのリリース以降にリリースされたプラットフォームのバージョンでのみ使用可能であるため、以前のプラットフォームバージョンを使用している場合は、現在の環境に更新してください。
- Elastic Beanstalk インスタンスプロファイルに AWSElasticBeanstalkWebTier またはr AWSElasticBeanstalkWorkerTier Elastic Beanstalk マネージドポリシーがない場合は、プロファイルに以下を追加して、この機能を有効にする必要があります。

  JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "logs:PutLogEvents",
      "logs:CreateLogStream"
    ],
    "Resource": [
    "*"
    ]
  }
  ]
}
```

Elastic Beanstalk が CloudWatch Logs を設定する方法

Elastic Beanstalk は、作成する各インスタンスにデフォルト設定を使用して CloudWatch Logs エージェントをインストールします。詳細については、『CloudWatch Logs エージェントリファレンス』を参照してください。

インスタンスログストリーミングを CloudWatch Logs に使用可能にすると、Elastic Beanstalk は環境のインスタンスからのログファイルを CloudWatch Logs に送信します。プラットフォームによって、ストリーミングされるログが異なります。次の表に、プラットフォーム別のログを示します。

プラットフォーム/プラットフォームブランチ	ログ
Docker/ プラットフォームブランチ: 64 ビット版 Amazon Linux 2 上で動作する Docker	• /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/docker • /var/log/docker-events.log • /var/log/eb-docker/containers/eb-current-app/stdouterr.log • /var/log/nginx/access.log • /var/log/nginx/error.log
Docker/ プラットフォームブランチ: 64 ビット版 Amazon Linux 2 上で動作する ECS	• /var/log/docker-events.log • /var/log/eb-ecs-mgr.log • /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/ecs/ecs-agent.log • /var/log/ecs/ecs-init.log
Go .NET Core on Linux Java/プラットフォームブランチ: 64 ビット版 Amazon Linux 2 上で動作する Corretto	• /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/web.stdout.log • /var/log/nginx/access.log • /var/log/nginx/error.log
Node.js Python	• /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/web.stdout.log • /var/log/httpd/access_log • /var/log/httpd/error_log • /var/log/nginx/access.log • /var/log/nginx/error.log
Tomcat	• /var/log/eb-engine.log

プラットフォーム/プラットフォームブランチ	ログ
PHP	• /var/log/eb-hooks.log • /var/log/httpd/access_log • /var/log/httpd/error_log • /var/log/nginx/access.log • /var/log/nginx/error.log
Windows Server の .NET	• C:\inetpub\logs\LogFiles\W3SVC1\u_ex*.log • C:\Program Files\Amazon\ElasticBeanstalk\logs\AWSDeployment.log • C:\Program Files\Amazon\ElasticBeanstalk\logs\Hooks.log
Ruby	• /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/puma/puma.log • /var/log/web.stdout.log • /var/log/nginx/access.log • /var/log/nginx/error.log

Amazon Linux AMI プラットフォーム上のログファイル

> **Note**
>
> 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。現在および完全にサポートされている Amazon Linux 2023 プラットフォームブランチへの移行の詳細については、「Elastic Beanstalk Linux アプリケーションを Amazon Linux 2023 または Amazon Linux 2 に移行する」を参照してください。

次の表に、Amazon Linux AMI (Amazon Linux 2 以前) に基づいて、プラットフォームブランチ上のインスタンスからストリーミングされるログファイルをプラットフォーム別に示します。

プラットフォーム/プラットフォームブランチ	ログ
Docker/ プラットフォームブランチ: 64 ビット版 Amazon Linux 上で動作する Docker	• /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/docker-events.log • /var/log/docker • /var/log/nginx/access.log • /var/log/eb-docker/containers/eb-current-app/stdouterr.log
Docker/ プラットフォームブランチ: 64 ビット版 Amazon Linux で動作するマルチコンテナ Docker	• /var/log/eb-activity.log • /var/log/ecs/ecs-init.log • /var/log/eb-ecs-mgr.log • /var/log/ecs/ecs-agent.log • /var/log/docker-events.log
Glassfish (Preconfigured Docker)	• /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/docker-events.log • /var/log/docker • /var/log/nginx/access.log
Go	• /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/nginx/access.log
Java/ プラットフォームブランチ: 64 ビット版 Amazon Linux 上で動作する Java 8 プラットフォームブランチ: 64 ビット版 Amazon Linux 上で動作する Java 7	• /var/log/eb-activity.log • /var/log/nginx/access.log • /var/log/nginx/error.log • /var/log/web-1.error.log • /var/log/web-1.log

プラットフォーム/プラットフォームブランチ	ログ
Tomcat	• /var/log/eb-activity.log • /var/log/httpd/error_log • /var/log/httpd/access_log • /var/log/nginx/error_log • /var/log/nginx/access_log
Node.js	• /var/log/eb-activity.log • /var/log/nodejs/nodejs.log • /var/log/nginx/error.log • /var/log/nginx/access.log • /var/log/httpd/error.log • /var/log/httpd/access.log
PHP	• /var/log/eb-activity.log • /var/log/httpd/error_log • /var/log/httpd/access_log
Python	• /var/log/eb-activity.log • /var/log/httpd/error_log • /var/log/httpd/access_log • /opt/python/log/supervisord.log
Ruby/ プラットフォームブランチ: 64 ビット版 Amazon Linux 上で動作する Puma with Ruby	• /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/puma/puma.log • /var/log/nginx/access.log

プラットフォーム/プラットフォームブランチ	ログ
Ruby/ プラットフォームブランチ: 64 ビット版 Amazon Linux 上で動作する Passenger with Ruby	• /var/log/eb-activity.log • /var/app/support/logs/passenger.log • /var/app/support/logs/access.log • /var/app/support/logs/error.log

Elastic Beanstalk は、ストリームするさまざまなログファイルに対して、CloudWatch Logs 内のロググループを構成します。CloudWatch Logs から特定のログファイルを検索するには、対応するロググループの名前を知る必要があります。ロググループの命名方式は、プラットフォームのオペレーティングシステムによって異なります。

Linux プラットフォームの場合は、インスタンス上のログファイルの場所に `/aws/elasticbeanstalk/`*`environment_name`* というプレフィックスを付けてロググループ名を取得します。たとえば、ファイル `/var/log/nginx/error.log` を取得するには、ロググループ `/aws/elasticbeanstalk/`*`environment_name`*`/var/log/nginx/error.log` を指定します

Windows プラットフォームの場合は、各ログファイルに対応するロググループについて、次の表を参照してください。

インスタンス上のログ	ロググループ
`C:\Program Files\Amazon\ElasticBeanstalk\logs\AWSDeployment.log`	`/aws/elasticbeanstalk/<environment-name>/EBDeploy-Log`
`C:\Program Files\Amazon\ElasticBeanstalk\logs\Hooks.log`	`/aws/elasticbeanstalk/<environment-name>/EBHooks-Log`
`C:\inetpub\logs\LogFiles` (ディレクトリ全体)	`/aws/elasticbeanstalk/<environment-name>/IIS-Log`

CloudWatch Logs へのインスタンスログのストリーミング

Elastic Beanstalk コンソール、EB CLI、または設定オプションを使用して、CloudWatch Logs へのインスタンスログのストリーミングを使用可能にすることができます。

これを有効にする前に、CloudWatch Logs エージェントで使用するための IAM アクセス権限を設定します。環境に割り当てるインスタンスプロファイルに次のカスタムポリシーをアタッチできます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Elastic Beanstalk コンソールを使用したインスタンスログストリーミング

CloudWatch Logs にインスタンスログをストリーミングするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [Instance log streaming to CloudWatch Logs]:
   - [Log streaming] を有効にします。

- [Retention] でログを保存する日数を指定します。
- 環境が終了した後にログを保存するかどうかを決定する [Lifecycle] 設定を選択します。

6. ページの最下部で [適用] を選択し変更を保存します。

ログストリーミングが有効になったら、[ソフトウェア] カテゴリまたはページに戻り、[ロググループ] のリンクを検索します。CloudWatch コンソールでログを表示するには、このリンクをクリックします。

EB CLI を使用したインスタンスログのストリーミング

EB CLI を使用してインスタンスログストリーミングを CloudWatch Logs に有効にするには、eb logs コマンドを使用します。

```
$ eb logs --cloudwatch-logs enable
```

また、eb logs を使用して、CloudWatch Logs からログを取得することもできます。すべての環境のインスタンスログを取り出すことも、コマンドの多くのオプションを使用して検索するログのサブセットを指定することもできます。たとえば、次のコマンドは、環境の一連のインスタンスログを取得し、`.elasticbeanstalk/logs` ディレクトリに保存します。

```
$ eb logs --all
```

特に、`--log-group` オプションを使用すると、インスタンス上の特定のログファイルに対応する特定のロググループのインスタンスログを取得できます。そのためには、取得するログファイルに対応するロググループの名前を知る必要があります。この情報は、「Elastic Beanstalk が CloudWatch Logs を設定する方法」にあります。

設定ファイルを使用したインスタンスログのストリーミング

環境を作成または更新する場合は、設定ファイルを使用して、CloudWatch Logs にインスタンスログストリーミングをセットアップして設定することができます。次の設定ファイルの例では、デフォルトのインスタンスログのストリーミングを有効にしています。Elastic Beanstalk は、環境のプラットフォーム用のログファイルのデフォルトセットをストリーミングします。例を使用するには、アプリケーションソースバンドルの最上位にある `.ebextensions` ディレクトリで、`.config` 拡張子を持つファイルにテキストをコピーします。

```
option_settings:
  - namespace: aws:elasticbeanstalk:cloudwatch:logs
```

```
    option_name: StreamLogs
    value: true
```

カスタムログファイルのストリーミング

CloudWatch Logs と Elastic Beanstalk の統合では、アプリケーションが生成するカスタムログファイルのストリーミングは直接サポートされません。カスタムログをストリーミングするには、設定ファイルを使用して直接 CloudWatch エージェントをインストールし、ファイルがプッシュされるよう設定します。設定ファイルの例については、「`logs-streamtocloudwatch-linux.config`」を参照してください。

> **Note**
>
> この例は、Windows プラットフォームでは動作しません。

CloudWatch Logs の設定の詳細については、「Amazon CloudWatch ユーザーガイド」の「CloudWatch エージェント設定ファイルリファレンス」を参照してください。

CloudWatch Logs 統合のトラブルシューティング

環境インスタンスログが見つかりません

CloudWatch Logs で予想される環境のインスタンスログの一部が見つからない場合は、次の一般的な問題を調査します。

- IAM ロールが必要な IAM アクセス許可を持っていません。
- CloudWatch Logs をサポート AWS リージョン していない で環境を起動しました。
- カスタムログファイルの 1 つが、指定されたパスに存在しません。

アプリケーションログが欠落または断続的

Elastic Beanstalk アプリケーションログ (`/var/log/web.stdout.log`) が欠落しているか断続的であるように見える場合は、rsyslog と journald のデフォルトのレート制限設定が原因である可能性があります。レート制限を完全に無効にするとこの問題を解決できますが、ディスクの過剰な使用量、サービス拒否の可能性、予期しないログバースト時のシステムパフォーマンスの低下につながる可能性があるため、お勧めしません。代わりに、次の を使用してレート制限を調整できます`.ebextensions configuration`。この設定では、レート制限間隔を 600 秒に引き上げ、バースト制限を大きくして、適切なログ記録とシステム保護のバランスを取ります。

スロットリングの問題

多数のインスタンスを同時に起動する Elastic Beanstalk オペレーションが のようなメッセージを返す場合`Error: fail to create log stream: ThrottlingException: Rate exceeded`、CloudWatch API への呼び出しが多すぎてスロットリングされます。

スロットリングの問題を解決するには、次のいずれかのアクションを実行します。

- ローリングデプロイでより小さなバッチサイズを使用して、同時更新を減らします。
- CreateLogStream の AWS アカウントのトランザクション/秒 (TPS) 制限サービスクォータの引き上げをリクエストします。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「CloudWatch Logs クォータ」とCloudWatch Logs サービスクォータの管理」を参照してください。 Amazon CloudWatch

Amazon CloudWatch Logs への Elastic Beanstalk 環境ヘルス情報のストリーミング

環境の拡張ヘルスレポートを有効にすると、CloudWatch Logs にヘルス情報をストリーミングするように環境を設定できます。このストリーミングは、Amazon EC2 インスタンスログのストリーミングとは独立しています。このトピックでは、環境ヘルス情報のストリーミングについて説明します。インスタンスログのストリーミングの詳細については、「Amazon CloudWatch Logs で Elastic Beanstalk を使用する」を参照してください。

環境のヘルスステータスのストリーミングを設定すると、Elastic Beanstalk は環境ヘルスの CloudWatch Logs ロググループを作成します。ロググループの名前名は `/aws/elasticbeanstalk/`*`environment-name`*`/environment-health.log` と呼ばれます。このロググループ内で、Elastic Beanstalk は *`YYYY-MM-DD`*`#`*`<hash-suffix>`* という名前のログストリームを作成します (日付ごとに複数のログストリームが存在する可能性があります)。

環境の状態が変化すると、Elastic Beanstalk はヘルスログストリームにレコードを追加します。このレコードは、ヘルス状態の変化、つまり新しい状態と変化の原因の説明を表します。たとえば、ロードバランサに障害が発生しているため、環境のステータスが Severe (重大) に変わることがあります。拡張ヘルスステータスの説明については、「状態の色とステータス」を参照してください。

CloudWatch Logs への環境ヘルスストリーミングの前提条件

CloudWatch Logs への環境ヘルスストリーミングを有効にするには、次の条件を満たす必要があります。

- プラットフォーム - 拡張ヘルスレポートをサポートするプラットフォームのバージョンを使用している必要があります。
- アクセス権限 - お使いの環境のヘルス情報をストリーミングするには、特定のログ記録に関連するアクセス許可を Elastic Beanstalk に付与する必要があります。Elastic Beanstalk が環境用に作成したサービスロール、`aws-elasticbeanstalk-service-role`、またはアカウントのサービスにリンクされたロール、`AWSServiceRoleForElasticBeanstalk` を使用していない環境では、カスタムサービスロールに次のアクセス権限を必ず追加してください。

```
{
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:log-group:/aws/elasticbeanstalk/*:log-stream:*"
}
```

CloudWatch Logs への環境ヘルスログのストリーミング

Elastic Beanstalk コンソール、EB CLI、または設定オプションを使用して、CloudWatch Logs への環境ヘルスのストリーミングを使用可能にすることができます。

Elastic Beanstalk コンソールを使用した環境ヘルスログのストリーミング

環境ヘルスログを CloudWatch Logs にストリーミングするには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [モニタリング] 設定カテゴリで、[編集] を選択します。
5. [ヘルスレポート] で、[システム] が [Enhanced (拡張)] に設定されていることを確認してください。
6. [Health event streaming to CloudWatch Logs] の下で
   - [Log streaming] を有効にします。
   - [Retention] でログを保存する日数を指定します。

- 環境が終了した後にログを保存するかどうかを決定する [Lifecycle] 設定を選択します。

7. ページの最下部で [適用] を選択し変更を保存します。

ログストリーミングが有効になったら、[モニタリング] カテゴリまたはページに戻り、[ロググループ] のリンクを検索します。CloudWatch コンソールで環境ヘルスログを表示するには、このリンクをクリックします。

EB CLI を使用した環境ヘルスログのストリーミング

EB CLI を使用して環境ヘルスログストリーミングを CloudWatch Logs に有効にするには、eb logs コマンドを使用します。

```
$ eb logs --cloudwatch-logs enable --cloudwatch-log-source environment-health
```

また、eb logs を使用して、CloudWatch Logs からログを取得することもできます。たとえば、次のコマンドは、環境のすべてのヘルスログを取得し、`.elasticbeanstalk/logs` ディレクトリに保存します。

```
$ eb logs --all --cloudwatch-log-source environment-health
```

設定ファイルを使用した環境ヘルスログのストリーミング

環境を作成または更新する場合は、設定ファイルを使用して、CloudWatch Logs に環境ヘルスストリーミングをセットアップして設定することができます。以下の例を使用するには、アプリケーションソースバンドルの最上位にある `.config` ディレクトリに、`.ebextensions` 拡張子を持つファイルにテキストをコピーします。この例では、環境ヘルスログストリーミングを有効にし、環境終了後にログを保存し、30 日間保存するように Elastic Beanstalk を構成しています。

Example ヘルスストリーミング設定ファイル

```
###########################################################################
##  Sets up Elastic Beanstalk to stream environment health information
##  to Amazon CloudWatch Logs.
##  Works only for environments that have enhanced health reporting enabled.
###########################################################################

option_settings:
  aws:elasticbeanstalk:cloudwatch:logs:health:
```

```
    HealthStreamingEnabled: true
    ### Settings below this line are optional.
    # DeleteOnTerminate: Delete the log group when the environment is
    # terminated. Default is false. If false, the health data is kept
    # RetentionInDays days.
    DeleteOnTerminate: false
    # RetentionInDays: The number of days to keep the archived health data
    # before it expires, if DeleteOnTerminate isn't set. Default is 7 days.
    RetentionInDays: 30
```

オプションのデフォルト値と有効な値については、aws:elasticbeanstalk:cloudwatch:logs:health を参照してください。

Amazon EventBridge で Elastic Beanstalk を使用する

Amazon EventBridge を使用すると、Elastic Beanstalk リソースをモニタリングし、他の AWS サービスを使用するターゲットアクションを開始するイベント駆動型ルールを設定できます。たとえば、実稼働環境のヘルスステータスが Warning に変わるたびに、それを Amazon SNS トピックに伝達することで、E メール通知を送信させるルールを設定できます。または、環境のヘルスステータスが Degraded または Severe 状態に変わった場合に通知を Slack に渡すように、Lambda 関数を設定することができます。

Amazon EventBridge でルールを作成して、次の Elastic Beanstalk イベントのいずれかに反応させることが可能です。

- 環境操作 (作成、更新、終了の各操作を含む) での状態の変化。このイベントは、状態の変更が開始された、成功した、あるいは失敗したことを示します。
- 他のリソースの状態の変更。モニタリングされる他のリソースには、環境に加えて、ロードバランサー、Auto Scaling グループ、インスタンスなどがあります。
- 環境のヘルスの移行。このイベントは、環境のヘルスステータスが特定の状態から別の状態に移行したことを示します。
- マネージド更新の状態の変更。このイベントは、状態の変更が開始された、成功した、あるいは失敗したことを示します。

関心のある特定の Elastic Beanstalk イベントをキャプチャするには、そのイベント固有のパターンを定義し、イベントを検出する EventBridge がそれを照合できるようにます。イベントパターンは、一致するイベントと同じ構造をしています。イベントのパターンでは、照合する対象のフィールドを引用符で囲み、検出したい値を指定します。イベントはベストエフォートベースで発生します。通

常の運用状況下では、Elastic Beanstalk から EventBridge にほぼリアルタイムで配信されます。ただし、イベントの配信を遅らせたり妨げたりする状況が発生する場合もあります。

Elastic Beanstalk イベントに含まれるフィールドと、そこで使用される可能性のある文字列値のリストについては、「Elastic Beanstalk イベントフィールドマッピング」を参照してください。EventBridge ルールがイベントパターンでどのように機能するかについては、「EventBridge のイベントとイベントパターン」を参照してください。

EventBridge を使用した Elastic Beanstalk リソースの監視

EventBridge を使用してルールを作成すると、Elastic Beanstalk がリソースに対してイベントを発生したときに実行されるアクションを定義できます。たとえば、環境のステータスが変化するたびに E メールメッセージを送信するルールを作成できます。

EventBridge コンソールには、Elastic Beanstalk イベントパターンを作成するために、事前定義済みのパターンがオプションとして用意されています。ルールの作成時に EventBridge コンソールでこのオプションを選択すると、Elastic Beanstalk イベントパターンをすばやく作成することが可能です。ユーザーが必要なのは、イベントフィールドと値の選択のみです。選択を行うと、コンソールによってイベントパターンが作成され、表示されます。また、作成されたイベントパターンを手動で編集し、カスタムパターンとして保存することもできます。コンソールには、詳細なサンプルイベントも表示されるので、これをコピーして、作成中のイベントパターンに貼り付けることもできます。

EventBridge コンソールにイベントパターンを入力またはコピーする場合は、コンソールの [カスタムパターン] オプションを選択します。これにより、前述したフィールドと値を選択する手順を実行する必要がなくなります。このトピックでは、イベントマッチングパターンと Elastic Beanstalk イベントの両方の例をご紹介します。

リソースイベントのルールを作成するには

1. EventBridge と Elastic Beanstalk を使用するアクセス許可を持つアカウント AWS を使用して にログインします。
2. Amazon EventBridge コンソール (https://console.aws.amazon.com/events/) を開きます。
3. ナビゲーションペインで ルール] を選択します。
4. [Create rule] を選択します。
5. ルールの [Name (名前)] を入力し、必要に応じて説明を入力します。
6. [イベントバス] として、[デフォルト] を選択してください。アカウントの AWS サービスがイベントを発行すると、常にアカウントのデフォルトのイベントバスに送られます。
7. [ルールタイプ] で、[イベントパターンを持つルール] を選択してください。

8. [Next (次へ)] を選択します。
9. [Event source] (イベントソース) で、[AWS events or EventBridge partner events] (イベントまたは EventBridge パートナーイベント) を選択してください。
10. (オプション)イベント例で、AWS イベントを選択します。検索フィールドに Elastic Beanstalk と入力します。これにより、表示するサンプルの Elastic Beanstalk イベントのリストが表示されます。この手順では、参照可能なサンプルイベントが表示されるだけです。ルール作成の結果には影響しません。Elastic Beanstalk イベントの例 セクションでは、このトピックで後述する、同じタイプのイベントの例について説明します。
11. [Event pattern] (イベントパターン) セクションで [Event pattern form] (イベントパターンフォーム) を選択します。

> **Note**
>
> すでにイベントパターンを定義するテキストがあり、EventBridge コンソールを使用してイベントパターンを作成する必要がない場合は、[カスタマーパターン (JSON エディタ)] を選択できます。その後、[イベントパターン] ボックスに対し、テキストを手動で入力するかコピーして貼り付けることができます。[次] を選択し、ターゲットの入力に関するステップに進みます。

12. [イベントソース] で、[AWS のサービス] を選択してください。
13. [AWS サービス] で [Elastic Beanstalk] を選択します。
14. [イベントタイプ] で [ステータスの変更] を選択します。
15. このステップでは、Elastic Beanstalk のイベントフィールドにある詳細タイプ、ステータス、重大度の処理方法について説明します。これらのフィールドと、そこで照合する値を選択することで、コンソールによってイベントパターンが作成され表示されます。

    - [特定の詳細タイプ] で値を 1 つだけ選択した場合は、階層内の次のフィールドで 1 つまたは複数の値を選択できます。
    - [特定の詳細タイプ] で値を複数選択した場合は、階層内の次のフィールドで特定の値を選択できません。イベントパターン内のフィールド間でマッチングロジックがあいまいになることを防ぐことができます。

    [環境] イベントフィールドはこの階層の影響を受けないため、次の手順で説明するように表示されます。

16. [環境] で、[任意の環境] または [特定の環境] を選択します。

- [特定の環境] を選択した場合は、ドロップダウンリストから 1 つまたは複数の環境を選択できます。EventBridge は、イベントパターンの [詳細] セクション の EnvironmentName[ ] リスト内で選択された、すべての環境を追加します。その後で、すべてのイベントがルールによりフィルタリングされ、選択された特定の環境のみが含められます。
- [任意の環境] を選択した場合、環境はイベントパターンに追加されません。このため、ルールは Elastic Beanstalk イベントを、環境に基づいてフィルタリングしません。

17. [Next (次へ)] を選択します。
18. [ターゲットタイプ] で、[AWS のサービス] を選択します。
19. [Select targets (ターゲットの選択)] で、Elastic Beanstalk からリソース状態変更イベントを受け取ったときに実行するターゲットアクションを選択します。

    たとえば、Amazon Simple Notification Service (SNS) トピックを使用して、イベントが発生したときに E メールまたはテキストメッセージを送信できます。これを行うには、Amazon SNS コンソールを使用して Amazon SNS トピックを作成する必要があります。詳細については、「ユーザー通知に Amazon SNS を使用する」を参照してください。

    > **Important**
    >
    > 一部のターゲットアクションでは、Amazon SNS や Lambda サービスなど他のサービスの使用が必要となり、追加料金が発生する可能性があります。 AWS 料金の詳細については、「」を参照してくださいhttps://aws.amazon.com/pricing/。一部の サービスは AWS 無料利用枠の一部です。新規のお客様は、無料でこれらのサービスをテストできる場合があります。詳細については「https://aws.amazon.com/free/」を参照してください。

20. (オプション) [他のターゲットを追加] を選択し、イベントルールに対し追加のターゲットアクションを指定します。
21. [Next (次へ)] を選択します。
22. (オプション) ルールに 1 つ以上のタグを入力します。詳細については、Amazon EventBridge ユーザーガイドのAmazon EventBridge のタグを参照してください。
23. 次へをクリックします。
24. ルールの詳細を確認し、ルールの作成 を選択します。

Elastic Beanstalk イベントパターンの例

イベントパターンは、一致するイベントと同じ構造をしています。イベントのパターンでは、照合する対象のフィールドを引用符で囲み、検出したい値を指定します。

- すべての環境のヘルスステータスの変更

```
{
   "source": [
    "aws.elasticbeanstalk"
  ],
  "detail-type": [
    "Health status change"
    ]
}
```

- 次の環境でのヘルスステータスの変更: `myEnvironment1` および `myEnvironment2` このイベントパターンは、これら 2 つの特定の環境にフィルタリングしますが、前出のフィルター処理をしていないヘルスステータス変更の例では、すべての環境のイベントを送信しています。

```
{"source": [
    "aws.elasticbeanstalk"
    ],
    "detail-type": [
        "Health status change"
    ],
    "detail": {
        "EnvironmentName": [
            "myEnvironment1",
            "myEnvironment2"
        ]
    }
}
```

- すべての環境での Elastic Beanstalk リソースステータスの変更

```
{
  "source": [
    "aws.elasticbeanstalk"
  ],
  "detail-type": [
    "Elastic Beanstalk resource status change"
```

```
    ]
}
```

- Elastic Beanstalk リソースのステータスが、次の環境で `Status` 環境の更新に失敗し、`Severity` エラーで変更されました: `myEnvironment1` および `myEnvironment2`

```
{"source": [
    "aws.elasticbeanstalk"
    ],
    "detail-type": [
        "Elastic Beanstalk resource status change"
    ],
    "detail": {
        "Status": [
            "Environment update failed"
            ],
        "Severity": [
            "ERROR"
            ],
        "EnvironmentName": [
            "myEnvironment1",
            "myEnvironment2"
        ]
    }
}
```

- ロードバランサー、Auto Scaling グループ、インスタンスによる、その他のリソースステータスの変更

```
{
   "source": [
    "aws.elasticbeanstalk"
  ],
  "detail-type": [
    "Other resource status change"
    ]
}
```

- すべての環境での、マネージド更新ステータスの変更

```
{
   "source": [
    "aws.elasticbeanstalk"
```

```
  ],
  "detail-type": [
    "Managed update status change"
    ]
}
```

- Elastic Beanstalk からすべてのイベントをキャプチャするには (`detail-type` セクションを除く)

```
{
  "source": [
    "aws.elasticbeanstalk"
  ]
}
```

Elastic Beanstalk イベントの例

リソースステータスの変更に対する Elastic Beanstalk イベントの例を次に示します。

```
{
   "version":"0",
   "id":"1234a678-1b23-c123-12fd3f456e78",
   "detail-type":"Elastic Beanstalk resource status change",
   "source":"aws.elasticbeanstalk",
   "account":"111122223333",
   "time":"2020-11-03T00:31:54Z",
   "region":"us-east-1",
   "resources":[
      "arn:was:elasticbeanstalk:us-east-1:111122223333:environment/myApplication/
myEnvironment"
   ],
   "detail":{
      "Status":"Environment creation started",
      "EventDate":1604363513951,
      "ApplicationName":"myApplication",
      "Message":"createEnvironment is starting.",
      "EnvironmentName":"myEnvironment",
      "Severity":"INFO"
   }
}
```

ヘルスステータスの変更に対する Elastic Beanstalk イベントの例を次に示します。

```
{
   "version":"0",
   "id":"1234a678-1b23-c123-12fd3f456e78",
   "detail-type":"Health status change",
   "source":"aws.elasticbeanstalk",
   "account":"111122223333",
   "time":"2020-11-03T00:34:48Z",
   "region":"us-east-1",
   "resources":[
      "arn:was:elasticbeanstalk:us-east-1:111122223333:environment/myApplication/
myEnvironment"
   ],
   "detail":{
      "Status":"Environment health changed",
      "EventDate":1604363687870,
      "ApplicationName":"myApplication",
      "Message":"Environment health has transitioned from Pending to Ok. Initialization
 completed 1 second ago and took 2 minutes.",
      "EnvironmentName":"myEnvironment",
      "Severity":"INFO"
   }
}
```

Elastic Beanstalk イベントフィールドマッピング

次の表は、Elastic Beanstalk イベントフィールドおよびそれに使用可能な文字列値と EventBridge `detail-type` フィールドのマッピングを示しています。EventBridge が、サービスのイベントパターンをどのように使用するかの詳細については、「EventBridge のイベントとイベントパターン」を参照してください。

EventBridge フィールドの detail-type	Elastic Beanstalk フィールドのステータス	Elastic Beanstalk フィールドの重大度	Elastic Beanstalk フィールドのメッセージ
Elastic Beanstalk リソーススス	環境の作成を開始しました	INFO	createEnvironment を開始しています。

EventBridge フィールドの detail-type	Elastic Beanstalk フィールドのステータス	Elastic Beanstalk フィールドの重大度	Elastic Beanstalk フィールドのメッセージ
テータスの変更	環境の作成に成功しました	INFO	createEnvironment が正常に完了しました。
	環境の作成に成功しました	INFO	環境: <環境名> を起動しました。ただし、起動中に問題が発生しました。詳細は、イベントログを参照してください。
	環境の作成に失敗しました	ERROR	環境に起動に失敗しました。
	環境の更新を開始しました	INFO	環境の更新を開始しています。
	環境の更新に成功しました	INFO	環境の更新が正常に完了しました。
	環境の更新に失敗しました	ERROR	設定のデプロイに失敗しました。
	環境の終了を開始しました	INFO	terminateEnvironment を開始しています。
	環境が正常に終了しました	INFO	terminateEnvironment が正常に完了しました。

EventBridge フィールドの detail-type	Elastic Beanstalk フィールドのステータス	Elastic Beanstalk フィールドの重大度	Elastic Beanstalk フィールドのメッセージ
	環境の終了に失敗しました	INFO	少なくとも 1 つの環境終了ワークフローが失敗したため、環境終了ステップが失敗しました。
その他のリソースステータスの変更	Auto Scaling グループが作成されました	INFO	createEnvironment を開始しています。
	Auto Scaling グループが削除されました	INFO	createEnvironment を開始しています。
	インスタンスが追加されました	INFO	ご使用の環境にインスタンス [i-123456789a12b1234] を追加しました。
	インスタンスが削除されました	INFO	ご使用の環境からインスタンス [i-123456789a12b1234] を削除しました。
	ロードバランサーが作成されました	INFO	作成されたロードバランサーの名前: <LB 名>

EventBridge フィールドの detail-type	Elastic Beanstalk フィールドのステータス	Elastic Beanstalk フィールドの重大度	Elastic Beanstalk フィールドのメッセージ
	ロードバランサーが削除されました	INFO	削除されたロードバランサーの名前: <LB 名>
ヘルスステータスの変更	環境ヘルスが変更されました	INFO/WARN	環境ヘルスが <healthStatus> に移行しました。
	環境ヘルスが変更されました	INFO/WARN	環境ヘルスが <healthStatus> から <healthStatus> に移行しました。
マネージド更新ステータスの変更	マネージド更新が開始されました	INFO	マネージドプラットフォーム更新が進行中です。
	マネージド更新に失敗しました	INFO	マネージド更新に失敗しました。%s 分後に再試行します。

を使用した Elastic Beanstalk リソースの検索と追跡 AWS Config

AWS Config は、 AWS アカウント内の AWS リソースの設定の詳細ビューを提供します。リソース間の関係、設定変更の履歴、関係と設定の時間的な変化を確認できます。を使用して AWS Config 、データコンプライアンスのリソース設定を評価するルールを定義できます。

いくつかの Elastic Beanstalk リソースタイプは、以下と統合されています AWS Config。

- アプリケーション
- アプリケーションバージョン

- 環境

次のセクションでは、これらのタイプのリソースを記録する AWS Config ように を設定する方法を示します。

詳細については AWS Config、「 AWS Config デベロッパーガイド」を参照してください。料金情報については、AWS Config 料金情報ページを参照してください。

セットアップ AWS Config

初期設定については AWS Config、 AWS Config デベロッパーガイドの以下のトピックを参照してください。

- コンソール AWS Config を使用した のセットアップ
- AWS Config を使用した のセットアップ AWS CLI

Elastic Beanstalk リソースを記録する AWS Config ように を設定する

デフォルトでは、 は、環境が実行されているリージョンで検出された、サポートされているすべてのタイプのリージョンリソースの設定変更 AWS Config を記録します。特定のリソースタイプの変更のみ、またはグローバルリソースの変更を記録する AWS Config ように をカスタマイズできます。

例えば、Elastic Beanstalk リソースの変更と、Elastic Beanstalk が開始する他の AWS リソースのサブセットを記録する AWS Config ように を設定できます。AWS Config コンソールを使用して、特定のタイプフィールドの AWS Config 「設定」ページのリソースとして Elastic Beanstalk を選択できます。そこから、記録する Elastic Beanstalk リソースタイプ (Application、ApplicationVersion、Environment) を選択できます。

次の図は、Application、ApplicationApplicationVersionVersion、Environment の Elastic Beanstalk リソースタイプの AWS Config 設定ページを示しています。

いくつかのリソースタイプを選択すると、[Specific types] リストが次のように表示されます。

リージョンリソースとグローバルリソース、および完全なカスタマイズ手順については、AWS Config 「記録するリソースの選択」を参照してください。

AWS Config コンソールでの Elastic Beanstalk 設定の詳細の表示

AWS Config コンソールを使用して Elastic Beanstalk リソースを検索し、その設定に関する現在および履歴の詳細を取得できます。次の例では Elastic Beanstalk 環境に関する情報を見つける方法を示します。

AWS Config コンソールで Elastic Beanstalk 環境を検索するには

1. AWS Config コンソールを開きます。
2. [リソース] を選択します。
3. [Resource (リソース)] のインベントリページで、[Resources (リソース)] を選択します。
4. [Resource type] メニューを開き、[ElasticBeanstalk] までスクロールし、1 つ以上の Elastic Beanstalk リソースタイプを選択します。

   > **Note**
   >
   > アプリケーション用に Elastic Beanstalk で作成した他のリソースタイプの設定の詳細を表示するには、その他のリソースタイプを選択します。たとえば、[EC2] の下の [インスタンス] を選択します。

5. [検索] を選択します。次の図の [2] を参照してください。

6. AWS Config 表示されるリソースのリストでリソース ID を選択します。

AWS Config は、選択したリソースに関する設定の詳細とその他の情報を表示します。

記録した設定の詳細全体を表示するには、[View Details (詳細を表示)] を選択します。

このページでリソースを検索して情報を表示する方法の詳細については、「 AWS Config デベロッパーガイド」の AWS 「リソース設定と履歴の表示」を参照してください。

AWS Config ルールを使用した Elastic Beanstalk リソースの評価

Elastic Beanstalk リソースの理想的な構成設定を表す AWS Config ルールを作成できます。事前定義された AWS マネージド Config ルールを使用するか、カスタムルールを定義できます。 は、リソー

スの設定の変更 AWS Config を継続的に追跡して、それらの変更がルールの条件に違反していないかどうかを判断します。 AWS Config コンソールには、ルールとリソースのコンプライアンスステータスが表示されます。

リソースがルールに違反し、非準拠としてフラグ付けされている場合、Amazon Simple Notification Service (Amazon SNS) トピックを使用してアラートを送信 AWS Config できます。これらの AWS Config アラートのデータをプログラムで消費するには、Amazon SNS トピックの通知エンドポイントとして Amazon Simple Queue Service (Amazon SQS) キューを使用します。 Amazon SNS 例えば、誰かが環境の Auto Scaling グループの設定を変更したときに、ワークフローを開始するようなコードを作成できます。

ルールの設定と使用の詳細については、「 AWS Config デベロッパーガイド」の AWS Config 「ルールを使用したリソースの評価」を参照してください。

Amazon DynamoDB で Elastic Beanstalk を使用する

Amazon DynamoDB は、フルマネージドの NoSQL データベースサービスであり、高速で予測可能なパフォーマンスとシームレスな拡張性が特長です。開発者が DynamoDB を使用して作成したデータベーステーブルでは、任意の量のデータを保存して取り出すことができ、どのような量のリクエストトラフィックも処理できます。DynamoDB では自動的に、テーブルのデータとトラフィックが多数のサーバーに分散されます。このとき、一貫性のある高速パフォーマンスを維持しながら、指定のリクエスト容量と保存されているデータ量を処理できるように、十分な数のサーバーが使用されます。すべてのデータ項目はソリッドステートドライブ (SSDs) に保存され、 AWS リージョン内の複数のアベイラビリティーゾーンに自動的にレプリケートされ、組み込みの高可用性とデータ耐久性を提供します。

ワーカー環境で定期的なタスクを使用する場合、Elastic Beanstalk は DynamoDB テーブルを作成し、これを使用してリーダーの選択を実行して、タスクに関する情報を保存します。環境内の各インスタンスは、数秒ごとにテーブルに書き込んでリーダーとなることを試み、スケジュールされるとタスクを実行します。

アプリケーション用の DynamoDB テーブルを作成するには、設定ファイルを使用します。設定ファイルを使用してテーブルを作成し、Node.js で AWS SDK for JavaScript を使用してテーブルに接続するサンプル Node.js アプリケーションについては、GitHub の eb-node-express-sample を参照してください。PHP での DynamoDB の使用例を示すチュートリアルについては、「」を参照してください例: DynamoDB、CloudWatch、SNS を使用する例については AWS SDK for Java、 AWS SDK for Java ドキュメントの「Manage Tomcat Session State with DynamoDB」を参照してください。

設定ファイルを使用して DynamoDB テーブルを作成した場合、テーブルは環境のライフサイクルに依存せず、環境を終了しても削除されません。個人情報が不必要に保持されないようにするには、不要なレコードを削除するか、テーブルを削除してください。

DynamoDB の詳細については、DynamoDB 開発者ガイドを参照してください。

Amazon ElastiCache で Elastic Beanstalk を使用するには

Amazon ElastiCache は、クラウド上でのインメモリ分散キャッシュ環境のセットアップ、管理、スケーリングを可能にするウェブサービスです。高パフォーマンス、拡張性、コストパフォーマンスに優れたインメモリキャッシュを提供するとともに、分散キャッシュ環境のデプロイと管理に伴う複雑さを解消します。ElastiCache は、Redis と Memcached のプロトコルに準拠しているため、既存の Redis と Memcached の環境でお客様が現在使用しているコード、アプリケーション、よく使用するツールは、このサービスでシームレスに機能します。ElastiCache の詳細については、Amazon ElastiCache の製品ページを参照してください。

Amazon ElastiCache で Elastic Beanstalk を使用するには

1. ElastiCache クラスターを作成します。
   - Redis を使用して ElastiCache クラスターを作成する方法については、「ElastiCache for Redis ユーザーガイド」の「Amazon ElastiCache for Redis の使用を開始する」を参照してください。
   - Memcached で ElastiCache クラスターを作成する方法については、「ElastiCache for Memcached ユーザーガイド」の「Amazon ElastiCache for Memcached の使用を開始する」を参照してください。
2. ElastiCache セキュリティグループを設定して、Elastic Beanstalk アプリケーションによって使用される Amazon EC2 セキュリティグループからのアクセスを許可します。 AWS マネジメントコンソールを使用して EC2 セキュリティグループの名前を検索する方法については、EC2 インスタンスドキュメントページのEC2 セキュリティグループ「」を参照してください。 EC2
   - Redis の詳細については、「ElastiCache for Redis ユーザーガイド」の「アクセスを承認する」を参照してください。
   - Memcached の詳細については、「ElastiCache for Memcached ユーザーガイド」の「アクセスを承認する」を参照してください。

設定ファイルを使用して Elastic Beanstalk 環境をカスタマイズすることにより、ElastiCache を使用できるようになります。ElastiCache を Elastic Beanstalk と統合する設定ファイルの例については、「例: ElastiCache」を参照してください。

Amazon Elastic File System で Elastic Beanstalk を使用する

Amazon Elastic File System (Amazon EFS) では、複数のアベイラビリティーゾーンのインスタンスにマウントできるネットワークファイルシステムを作成できます。Amazon EFS ファイルシステムは、セキュリティグループを使用して、デフォルトまたはカスタム VPC にあるネットワーク経由のアクセスを制御する AWS リソースです。

Elastic Beanstalk 環境では、Amazon EFS を使用して、ユーザーがアップロードまたは変更したアプリケーションのファイルを保存する共有ディレクトリを作成できます。アプリケーションは、ローカルストレージなど、マウントされた Amazon EFS ボリュームを処理できます。そうすれば、アプリケーションコードを変更して複数のインスタンスにスケールアップする必要がなくなります。

Amazon EFS の詳細については、Amazon Elastic File System ユーザーガイドを参照してください。

> **Note**
>
> Elastic Beanstalk は、Amazon EC2 インスタンスのアプリケーションディレクトリのオーナーとして設定できる webapp ユーザーを作成します。詳細については、このガイドの設計上の考慮事項トピックの永続的ストレージを参照してください。

セクション

- 設定ファイル
- 暗号化されたファイルシステム
- サンプルアプリケーション
- ファイルシステムのクリーンアップ

設定ファイル

Elastic Beanstalk は、Amazon EFS ファイルシステムを作成およびマウントするために使用できる設定ファイルを提供します。環境の一部としてAmazon EFS ボリュームを作成することも、作成したAmazon EF ボリュームを Elastic Beanstalk とは関係なくマウントすることもできます。

- storage-efs-createfilesystem.config – `Resources` キーを使用して、Amazon EFS で新しいファイルシステムとマウントポイントを作成します。環境のすべてのインスタンスは、共有されたスケーラブルなストレージのため、同じファイルシステムに接続できます。`storage-efs-mountfilesystem.config` を使用して、各インスタンスでファイルシステムをマウントできます。

> **内部リソース**
>
> 設定ファイルで作成するリソースは、環境のライフサイクルに関連付けられます。環境を終了するか、構成ファイルを削除すると、これらのリソースは失われます。

- storage-efs-mountfilesystem.config – 環境のインスタンスのローカルパスに、Amazon EFS ファイルシステムをマウントします。ボリュームは、`storage-efs-createfilesystem.config` を使用して環境の一部として作成することができます。または、Amazon EFS コンソール AWS CLI または AWS SDK を使用して環境にマウントすることもできます。

設定ファイルを使用するには、最初に `storage-efs-createfilesystem.config` を使用して Amazon EFS ファイルシステムを作成します。設定ファイルの指示に従って、ソースコードの .ebextensions ディレクトリにファイルを追加し、VPC でファイルシステムを作成します。

更新されたソースコードを Elastic Beanstalk 環境にデプロイします。これは、ファイルシステムが正常に作成されることを確認するためです。次に、`storage-efs-mountfilesystem.config` を追加し、環境のインスタンスにファイルシステムをマウントします。2 つの別のデプロイでこの操作を行うことで、マウントオペレーションに失敗した場合でも、ファイルシステムはそのまま残ります。同じデプロイで両方の操作を行った場合、デプロイが失敗すると、いずれかのステップの問題によりファイルシステムは終了します。

暗号化されたファイルシステム

Amazon EFS は暗号化されたファイルシステムをサポートします。このトピックで説明されている `storage-efs-createfilesystem.config` 設定ファイルは、2 つのカスタムオプションを定義します。これらのオプションを使用して、Amazon EFS の暗号化ファイルシステムを作成できます。詳細については、設定ファイルの手順を参照してください。

サンプルアプリケーション

Elastic Beanstalk では、共有ストレージ用に Amazon EFS を使用するサンプルアプリケーションも提供しています。2 つのプロジェクトには、標準の WordPress または Drupal インストーラーととも

に使用して、ロードバランシングされた環境でブログまたは他のコンテンツ管理システムを実行できる設定ファイルがあります。ユーザーが写真またはその他のメディアをアップロードすると、ファイルは Amazon EFS ファイルシステムに保存されます。これにより、Amazon S3 にアップロードされたファイルを保存するためにプラグインを使用する代替手段を使用する必要がなくなります。

- ロードバランシングされた WordPress – これには、WordPress を安全にインストールし、ロードバランシングされた Elastic Beanstalk 環境で実行するための設定ファイルが含まれます。
- ロードバランシングされた Drupal – これには、Drupal を安全にインストールし、ロードバランシングされた Elastic Beanstalk 環境で実行するための設定ファイルと手順が含まれます。

ファイルシステムのクリーンアップ

Elastic Beanstalk 環境の一部として構成ファイルを使用する Amazon EFS ファイルシステムを作成した場合、Elastic Beanstalk は、環境を終了するときにファイルシステムを削除します。実行中のアプリケーションのストレージコストを最小限に抑えるには、アプリケーションが必要としないファイルを定期的に削除します。または、アプリケーションコードがファイルのライフサイクルを正しく維持していることを確認します。

> **Important**
>
> Elastic Beanstalk 環境外で Amazon EFS ファイルシステムを作成し、環境のインスタンスにマウントした場合、環境を終了するときに Elastic Beanstalk はファイルシステムを削除しません。個人情報が保持されないようにし、ストレージコストを回避するためには、アプリケーションが保存したファイルが不要になった場合削除します。または、ファイルシステム全体を削除することもできます。

での Elastic Beanstalk の使用 AWS Identity and Access Management

AWS Identity and Access Management (IAM) は、 AWS リソースへのアクセスを安全に制御するのに役立ちます。このセクションには、IAM ポリシー、インスタンスプロファイル、サービスロールを使用するためのリファレンスが含まれています。

アクセス許可の概要については、「Elastic Beanstalk サービスロール、インスタンスプロファイル、ユーザーポリシー」を参照してください。ほとんどの環境では、お客様の最初の環境を起動したときに Elastic Beanstalk コンソールで作成するように求められるサービスロールとインスタンスプロ

ファイルに、必要なすべてのアクセス許可があります。同様に、フルアクセスと読み取り専用アクセス用に Elastic Beanstalk によって提供される管理ポリシーには、日常の使用に必要なユーザーアクセス許可がすべて含まれています。

IAM ユーザーガイドには、アクセス AWS 許可の詳細が記載されています。

トピック

- Elastic Beanstalk インスタンスプロファイルの管理
- Elastic Beanstalk サービスロールの管理
- Elastic Beanstalk でのサービスにリンクされたロールの使用
- Elastic Beanstalk ユーザーポリシーの管理
- Elastic Beanstalk の Amazon リソースネームの形式
- Elastic Beanstalk アクションのリソースと条件
- タグを使用した Elastic Beanstalk リソースへのアクセスのコントロール
- 管理ポリシーに基づくポリシーの例
- リソースに対するアクセス許可に基づいたポリシーの例
- 環境間の Amazon S3 バケットアクセスの防止

Elastic Beanstalk インスタンスプロファイルの管理

インスタンスプロファイルは、インスタンスの起動時にロール情報を Amazon EC2 インスタンスに渡すために使用できる AWS Identity and Access Management (IAM) ロールのコンテナです。

AWS アカウントに EC2 インスタンスプロファイルがない場合は、IAM サービスを使用して作成する必要があります。その後、作成する新しい環境に EC2 インスタンスプロファイルを割り当てることができます。Elastic Beanstalk コンソールの環境の作成ステップでは、IAM コンソールにアクセスできるため、必要なアクセス許可を持つ EC2 インスタンスプロファイルを作成できます。

> **Note**
>
> 以前は、Elastic Beanstalk は、 AWS アカウント`aws-elasticbeanstalk-ec2-role`が初めて環境を作成したときに、 という名前のデフォルトの EC2 インスタンスプロファイルを作成していました。このインスタンスプロファイルには、デフォルトの管理ポリシーが含まれていました。アカウントに既にこのインスタンスプロファイルがある場合、引き続き環境に割り当てることができます。

ただし、最近の AWS セキュリティガイドラインでは、 AWS サービスが信頼ポリシーを使用して他の AWS サービスである EC2 へのロールを自動的に作成することはできません。これらのセキュリティガイドラインにより、Elastic Beanstalk はデフォルトの `aws-elasticbeanstalk-ec2-role` インスタンスプロファイルを作成しなくなりました。

マネージドポリシー

Elastic Beanstalk には、環境がさまざまなユースケースに対応できるように、いくつかの管理ポリシーが用意されています。環境のデフォルトユースケースを満たすには、これらのポリシーを EC2 インスタンスプロファイルのロールにアタッチする必要があります。

- AWSElasticBeanstalkWebTier – ログを Amazon S3 にアップロードし、デバッグ情報を AWS X-Rayにアップロードするアクセス許可をアプリケーションに付与します。管理ポリシーの内容を表示するには、「AWS 管理ポリシーリファレンスガイド」の「AWSElasticBeanstalkWebTier」を参照してください。
- AWSElasticBeanstalkWorkerTier - ログのアップロード、デバッグ、メトリクスのパブリッシュ、ワーカーインスタンスタスク (キュー管理やリーダー選択、定期的なタスクなど) に対するアクセス許可を付与します。管理ポリシーの内容を表示するには、「AWS 管理ポリシーリファレンスガイド」の「AWSElasticBeanstalkWorkerTier」を参照してください。
- AWSElasticBeanstalkMulticontainerDocker - Docker 環境のクラスタータスクを調整する許可を Amazon Elastic Container Service に付与します。管理ポリシーの内容を表示するには、「AWS 管理ポリシーリファレンスガイド」の「AWSElasticBeanstalkMulticontainerDocker」を参照してください。

Important

Elastic Beanstalk マネージド型ポリシーは、詳細なアクセス権限を提供しません。Elastic Beanstalk アプリケーションの操作に必要となる可能性のある、すべてのアクセス権限が付与されます。場合によっては、管理ポリシーのアクセス許可をさらに制限することもできます。1 つのユースケースの例については、「環境間の Amazon S3 バケットアクセスの防止」を参照してください。

また、当社の管理ポリシーでは、Elastic Beanstalk では管理されておらず、お客様によりソリューションに追加されるような、カスタムリソースのためのアクセス許可についてもサポートしていません。よりきめの細かなアクセス許可、必要最小限のアクセス許可、または

リソースに対するアクセス許可をカスタムで作成するには、カスタムポリシーを使用します。

EC2 の信頼関係ポリシー

環境内の EC2 インスタンスが必要なロールを引き受けるには、インスタンスプロファイルは、次のように信頼関係ポリシーで Amazon EC2 を信頼されたエンティティとして指定する必要があります。

JSON

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

アクセス権限をカスタマイズするには、デフォルトのインスタンスプロファイルにアタッチされたロールにポリシーを追加するか、権限が限定された独自のインスタンスプロファイルを作成します。

セクション

- インスタンスプロファイルの作成
- デフォルトのインスタンスプロファイルにアクセス許可を追加する
- インスタンスプロファイルに割り当てられた許可の確認
- 情報が古くなったデフォルトのインスタンスプロファイルを更新する

インスタンスプロファイルの作成

インスタンスプロファイルは、EC2 インスタンスによるロールの引き受けを許可する標準 IAM ロールのラッパーです。デフォルトの Elastic Beanstalk 管理ポリシーを使用してインスタンスプロファイルを作成できます。追加のインスタンスプロファイルを作成して、さまざまなアプリケーションのアクセス許可をカスタマイズすることもできます。または、ワーカー階層または ECS マネージド Docker 環境のアクセス許可を付与する 2 つのマネージドポリシーを含まないインスタンスプロファイルを作成することもできます。

デフォルトの管理ポリシーを使用してインスタンスプロファイルを作成するには

1. IAM コンソールの [Roles (ロール)] ページを開きます。
2. [ロールの作成] を選択してください。
3. [信頼できるエンティティタイプ] で、[AWS サービス] を選択します。
4. サービスまたはユースケースで、Elastic Beanstalk を選択します。
5. ユースケースでは、Elastic Beanstalk – Compute を選択します。
6. [次へ] を選択します。
7. [Role name] (ロール名) に入力します。

   Elastic Beanstalk コンソールが提案するデフォルトロールの名前を入力できます: `aws-elasticbeanstalk-ec2-role`。
8. アクセス許可ポリシーに以下が含まれていることを確認し、次へを選択します。

   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`
9. [ロールの作成] を選択してください。

特定の選択した管理ポリシーを使用してインスタンスプロファイルを作成するには

1. IAM コンソールの [Roles (ロール)] ページを開きます。
2. [ロールの作成] を選択してください。
3. [信頼されたエンティティタイプ] から、[AWS サービス] を選択します。
4. [ユースケース] で、[EC2] を選択してください。
5. [次へ] を選択します。

6. Elastic Beanstalk により指定された適切な管理ポリシーや、アプリケーションで必要なアクセス許可を付与するその他のポリシーがあれば、それらもアタッチします。
7. [次へ] を選択します。
8. ロールの名前を入力します。
9. (オプション) ロールにタグを追加します。
10. [ロールの作成] を選択してください。

デフォルトのインスタンスプロファイルにアクセス許可を追加する

アプリケーションがデフォルトのインスタンスプロファイルでアクセス許可が付与されていない AWS APIs またはリソースにアクセスする場合は、IAM コンソールでアクセス許可を付与するポリシーを追加します。

デフォルトのインスタンスプロファイルにアタッチされたロールにポリシーを追加するには

1. IAM コンソールの [ロール] ページを開きます。
2. EC2 インスタンスプロファイルとして割り当てられたロールを選択します。
3. [Permissions (アクセス許可)] タブで、[Attach policy (ポリシーの添付)] を選択します。
4. アプリケーションで使用する追加サービスの管理ポリシーを選択します。例えば、`AmazonS3FullAccess`、`AmazonDynamoDBFullAccess` です。
5. Attach policy] (ポリシーのアタッチ) を選択してください。

インスタンスプロファイルに割り当てられた許可の確認

デフォルトのインスタンスプロファイルに割り当てられる権限は、作成日時、環境の最終起動日時、使用したクライアントにより異なります。デフォルトのインスタンスプロファイルに含まれるアクセス許可は IAM コンソールで確認できます。

デフォルトのインスタンスプロファイルのアクセス権限を確認する

1. IAM コンソールの [Roles (ロール)] ページを開きます。
2. EC2 インスタンスプロファイルとして割り当てられたロールを選択します。
3. [Permissions (アクセス許可)] タブで、ロールにアタッチされたポリシーのリストを確認します。
4. ポリシーにより付与されるアクセス権限を表示するには、ポリシーを選択します。

情報が古くなったデフォルトのインスタンスプロファイルを更新する

デフォルトのインスタンスプロファイルに必要な許可がない場合、EC2 インスタンスプロファイルとして割り当てられたロールに管理ポリシーを手動で追加できます。

デフォルトのインスタンスプロファイルにアタッチされたロールに管理ポリシーを追加するには

1. IAM コンソールの [Roles (ロール)] ページを開きます。
2. EC2 インスタンスプロファイルとして割り当てられたロールを選択します。
3. [Permissions (アクセス許可)] タブで、[Attach policy (ポリシーの添付)] を選択します。
4. ポリシーをフィルタリングするには、「**`AWSElasticBeanstalk`**」と入力します。
5. 次のポリシーを指定し、[Attach policy (ポリシーのアタッチ)] を選択します。
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`

Elastic Beanstalk サービスロールの管理

環境を管理およびモニタリングするために、 AWS Elastic Beanstalk はユーザーに代わって環境リソースに対してアクションを実行します。Elastic Beanstalk には、これらのアクションを実行するための特定のアクセス許可が必要であり、これらのアクセス許可を取得するために AWS Identity and Access Management (IAM) サービスロールを引き受けます。

Elastic Beanstalk は、サービスロールを引き受けるときはいつでも一時的セキュリティ認証情報を使用する必要があります。これらの認証情報を取得するために、Elastic Beanstalk により、リージョン固有エンドポイントの AWS Security Token Service (AWS STS) にリクエストが送信されます。詳細については、IAM ユーザーガイドの「IAM の一時的なセキュリティ認証情報」を参照してください。

> **Note**
>
> 環境が配置されているリージョンの AWS STS エンドポイントが非アクティブ化されている場合、Elastic Beanstalk は非アクティブ化できない代替エンドポイントにリクエストを送信します。このエンドポイントは別のリージョンに関連付けられています。つまり、このリクエストはクロスリージョンのリクエストとなります。詳細については、IAM ユーザーガイ

ドの「 AWS リージョン AWS STS でのアクティブ化と非アクティブ化」を参照してください。

Elastic Beanstalk コンソールと EB CLI を使用したサービスロールの管理

Elastic Beanstalk コンソールと EB CLI を使用すると、十分なアクセス許可セットを含むサービスロールを環境に設定できます。デフォルトのサービスロールが作成され、その中の管理ポリシーが使用されます。

マネージドサービスロールのポリシー

Elastic Beanstalk により、拡張ヘルスモニタリング用の管理ポリシーが 1 つ、またマネージドプラットフォーム更新用の (必要な他のアクセス許可を含む) 管理ポリシーが 1 つ、それぞれ指定されます。コンソールと EB CLI は、デフォルトのサービスロールを作成する際に、これらのポリシーの両方を割り当てます。これらのポリシーは、このデフォルトのサービスロールでのみ使用するものです。アカウント内の他のユーザーまたはロールでは使用しないでください。

`AWSElasticBeanstalkEnhancedHealth`

このポリシーでは、インスタンスおよび環境のヘルスをモニタリングするアクセス許可を Elastic Beanstalk に付与します。このポリシーには、Elastic Beanstalk にワーカー環境のキューアクティビティのモニタリングを許可する Amazon SQS アクションも含まれています。この管理ポリシーの内容を確認するには、AWS 「 管理ポリシーリファレンスガイド」のAWSElasticBeanstalkEnhancedHealth」ページを参照してください。

`AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy`

このポリシーでは、お客様に代わって環境を更新してマネージドプラットフォームを更新するアクセス許可を Elastic Beanstalk に付与します。この管理ポリシーの内容を確認するには、AWS 「 管理ポリシーリファレンスガイド」のAWSElasticBeanstalkManagedUpdatesCustomerRolePolicy」ページを参照してください。

サービスレベルでのアクセス許可のグループ化

このポリシーは、提供された一連の許可に基づくステートメントごとにグループ化されます。

- *`ElasticBeanstalkPermissions`* – このアクセス許可グループは、Elastic Beanstalk サービスアクション (Elastic Beanstalk API) を呼び出す際に使用します。

- *AllowPassRoleToElasticBeanstalkAndDownstreamServices* – このアクセス許可グループにより、Elastic Beanstalk や AWS CloudFormationなどのダウンストリームサービスに対し、任意のロールを渡すことが可能になります。
- *ReadOnlyPermissions* – このアクセス許可グループは、実行中の環境に関する情報を収集するため使用します。
- **OperationPermissions* – この命名パターンを持つグループは、プラットフォームの更新を実行するために必要なオペレーションを呼び出すためのものです。
- **BroadOperationPermissions* – この命名パターンを持つグループは、プラットフォームの更新を実行するために必要なオペレーションを呼び出すためのものです。またこれには、レガシー環境をサポートするための広範なアクセス許可も含まれています。
- **TagResource*– この命名パターンのグループは、tag-on-create API を使用して Elastic Beanstalk 環境で作成されているリソースにタグをアタッチする呼び出し用です。

管理ポリシーの内容は、IAM コンソールの [ポリシー] ページでも表示することができます。

> **Important**
>
> Elastic Beanstalk マネージド型ポリシーは、詳細なアクセス権限を提供しません。Elastic Beanstalk アプリケーションの操作に必要となる可能性のある、すべてのアクセス権限が付与されます。場合によっては、管理ポリシーのアクセス許可をさらに制限することもできます。1 つのユースケースの例については、「環境間の Amazon S3 バケットアクセスの防止」を参照してください。
> また、当社の管理ポリシーでは、Elastic Beanstalk では管理されておらず、お客様によりソリューションに追加されるような、カスタムリソースのためのアクセス許可についてもサポートしていません。よりきめの細かなアクセス許可、必要最小限のアクセス許可、またはリソースに対するアクセス許可をカスタムで作成するには、カスタムポリシーを使用します。

非推奨の マネージドポリシー

これまで Elastic Beanstalk では、AWSElasticBeanstalkService マネージドサービスロールポリシーをサポートしていました。現在このポリシーは、AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy に置き換えられています。以前のポリシーは、IAM コンソールから表示および使用できる場合があります。

管理ポリシーの内容を表示するには、「AWS 管理ポリシーリファレンスガイド」の「AWSElasticBeanstalkService」を参照してください。

ただし、新しい管理ポリシー (AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy) の使用を選択されることをお勧めします。カスタムリソースをご使用の場合は、カスタムポリシーを追加して、そのリソースにアクセス許可を付与します。

Elastic Beanstalk コンソールを開きます。

Elastic Beanstalk コンソールで環境を起動すると、`aws-elasticbeanstalk-service-role` という名前のデフォルトのサービスロールが作成され、デフォルトのアクセス許可を含む管理ポリシーが、そのサービスロールにアタッチされます。

Elastic Beanstalk が `aws-elasticbeanstalk-service-role` ロールを引き受けることができるようにするために、サービスロールは Elastic Beanstalk を信頼関係ポリシーの信頼されたエンティティとして指定します。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "",
        "Effect": "Allow",
        "Principal": {
          "Service": "elasticbeanstalk.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
            "sts:ExternalId": "elasticbeanstalk"
          }
        }
      }
    ]
}
```

環境に対してマネージドプラットフォーム更新を有効にすると、Elastic Beanstalk は、マネージド更新を実行するための個別のマネージド更新サービスロールを引き受けます。Elastic

Beanstalk コンソールのデフォルトでは、このマネージド更新サービスロールにも、生成された同じサービスロール (`aws-elasticbeanstalk-service-role`) が使用されます。デフォルトのサービスロールを変更すると、マネージド更新サービスにリンクされたロール (`AWSServiceRoleForElasticBeanstalkManagedUpdates`) を使用するように、コンソールによってマネージド更新サービスロールが設定されます。サービスにリンクされたロールの詳細については、「the section called “サービスにリンクされたロールの使用”」を参照してください。

> **Note**
>
> アクセス許可の状態によっては、Elastic Beanstalk サービスが、サービスにリンクされたロールの作成に必ず成功するとは限りません。このため、明示的な作成がコンソールで試行されます。サービスにリンクされたロールがアカウントに確実に付与されるようにするには、コンソールを使用して 1 回以上は環境を作成します。環境を作成する前に、マネージド更新が有効になるように設定する必要があります。

EB CLI の使用

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) の the section called “eb create” コマンドを使用して環境を起動する際に、`--service-role` オプションでサービスロールを指定しない場合は、Elastic Beanstalk によってデフォルトのサービスロール (`aws-elasticbeanstalk-service-role`) が作成されます。デフォルトのサービスロールが既に存在する場合、Elastic Beanstalk はそのサービスロールを新しい環境で使用します。また、Elastic Beanstalk コンソールでも、このような状況に対し同様のアクションが実行されます。

ただし、EB CLI コマンドオプションを使用する場合には、コンソールとは異なりマネージド更新サービスロールを指定することはできません。環境のマネージド更新を有効にする場合は、設定オプションを使用してマネージド更新サービスロールを設定する必要があります。次の例では、マネージド更新を有効にし、デフォルトのサービスロールをマネージド更新サービスロールとして使用しています。

Example .ebextensions/managed-platform-update.config

```
option_settings:
  aws:elasticbeanstalk:managedactions:
    ManagedActionsEnabled: true
    PreferredStartTime: "Tue:09:00"
    ServiceRoleForManagedUpdates: "aws-elasticbeanstalk-service-role"
  aws:elasticbeanstalk:managedactions:platformupdate:
```

```
    UpdateLevel: patch
    InstanceRefreshEnabled: true
```

Elastic Beanstalk API を使用したサービスロールの管理

Elastic Beanstalk API の `CreateEnvironment` アクションを使用して環境を作成する場合は、`ServiceRole` 名前空間の `aws:elasticbeanstalk:environment` 設定オプションを使用して、サービスロールを指定します。Elastic Beanstalk API を使用しての拡張ヘルスモニタリングの詳細については、「Elastic Beanstalk API での拡張ヘルスレポートの使用」を参照してください。

さらに、環境に対してマネージドプラットフォーム更新を有効にする場合は、`aws:elasticbeanstalk:managedactions` 名前空間の `ServiceRoleForManagedUpdates` オプションを使用してマネージド更新サービスロールを指定できます。

サービスにリンクされたロールの使用

サービスにリンクされたロールは、Elastic Beanstalk によって事前定義された一意のタイプのサービスロールであり、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ます。このサービスリンクロールはお客様のアカウントに関連付けられています。Elastic Beanstalk によって一度作成されたロールは、追加の環境を作成するときにも使用されます。Elastic Beanstalk 環境でのサービスにリンクされたロールの使用の詳細については、「Elastic Beanstalk でのサービスにリンクされたロールの使用」を参照してください。

Elastic Beanstalk API を使用して環境を作成する際にサービスロールを指定しない場合は、(それが存在しない場合には) モニタリングサービスにリンクされたロールが、Elastic Beanstalk によりアカウントに作成されます。Elastic Beanstalk は、作成された環境でこのロールを使用します。IAM を使用して、モニタリングサービスにリンクされたロールを、アカウント用として事前に作成することもできます。アカウントでこのロールの取得が完了したら、そのロールにより Elastic Beanstalk API、Elastic Beanstalk コンソール、または EB CLI を使用しての環境の作成が可能になります。

環境に対してマネージドプラットフォーム更新を有効にし、名前空間 `AWSServiceRoleForElasticBeanstalkManagedUpdates` の `ServiceRoleForManagedUpdates` オプションの値として `aws:elasticbeanstalk:managedactions` を指定した場合、(それが存在しない場合には) アカウントのためのマネージド更新サービスにリンクされたロールが、Elastic Beanstalk により作成されます。Elastic Beanstalk は、この作成されたロールを使用して、新しい環境でのマネージド更新を実行します。

> **Note**
>
> 環境の作成時に Elastic Beanstalk でモニタリングサービスおよびマネージド更新サービスにリンクされたロールをアカウントに作成する場合は、`iam:CreateServiceLinkedRole` アクセス許可が必要です。このアクセス許可がない場合、環境の作成は失敗し、問題を報告するメッセージが表示されます。
> 別の方法として、サービスにリンクされたロールを作成するアクセス許可を持つ別のユーザーが IAM を使用して、サービスにリンクされたロールを事前に作成できます。この方法では、環境を作成するために `iam:CreateServiceLinkedRole` のアクセス許可は必要ありません。

デフォルトのサービスロールのアクセス許可を確認する

デフォルトのサービスロールに付与されるアクセス許可は、作成日時、最後に環境を起動した日時、使用したクライアントに基づき変化します。デフォルトのサービスロールに付与されるアクセス許可は、IAM コンソールで確認できます。

デフォルトのサービスロールのアクセス権限を確認する

1. IAM コンソールで [ロール] ページを開きます。
2. [aws-elasticbeanstalk-service-role] を選択します。
3. [Permissions (アクセス許可)] タブで、ロールにアタッチされたポリシーのリストを確認します。
4. ポリシーにより付与されるアクセス権限を表示するには、ポリシーを選択します。

古くなったデフォルトのサービスロールを更新する

必要なアクセス許可がデフォルトのサービスロールに付与されていない場合は、Elastic Beanstalk 環境マネジメントコンソールで新しい環境を作成し、アクセス許可を更新します。

あるいは、デフォルトのサービスロールに管理ポリシーを手動で追加することも可能です。

デフォルトのサービスロールに管理ポリシーを追加する

1. IAM コンソールで [ロール] ページを開きます。
2. [aws-elasticbeanstalk-service-role] を選択します。
3. [Permissions (アクセス許可)] タブで、[Attach policy (ポリシーの添付)] を選択します。

4. **AWSElasticBeanstalk** を入力してポリシーをフィルタリングします。
5. 次のポリシーを指定し、[Attach policy (ポリシーのアタッチ)] を選択します。
   - `AWSElasticBeanstalkEnhancedHealth`
   - `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy`

デフォルトのサービスロールにアクセス許可を付与する

アプリケーションに、アクセス許可がデフォルトのサービスロールに含まれていない AWS リソースを参照する設定ファイルが含まれている場合、Elastic Beanstalk には追加のアクセス許可が必要になる場合があります。これらの追加のアクセス許可は、マネージド更新の実行中に構成ファイルを処理する際、必要な参照を解決するために必要です。アクセス許可がない場合には、更新が失敗し、Elastic Beanstalk からは、不足しているアクセス許可を示すメッセージが出力されます。IAM コンソールで次の手順を実行し、新たなサービスのためのアクセス許可を、デフォルトのサービスロールに追加します。

デフォルトのサービスロールにその他のポリシーを追加する

1. IAM コンソールで [ロール] ページを開きます。
2. [aws-elasticbeanstalk-service-role] を選択します。
3. [Permissions (アクセス許可)] タブで、[Attach policy (ポリシーの添付)] を選択します。
4. アプリケーションで使用する追加サービスの管理ポリシーを選択します。例えば、`AmazonAPIGatewayAdministrator`、`AmazonElasticFileSystemFullAccess` です。
5. Attach policy] (ポリシーのアタッチ) を選択してください。

サービスロールの作成

デフォルトのサービスロールを使用できない場合は別途サービスロールを作成します。

サービスロールを作成する

1. IAM コンソールで [ロール] ページを開きます。
2. [ロールの作成] を選択してください。
3. [AWS service] (サービス) で、[AWS Elastic Beanstalk] を選択してから、ユースケースを選択します。

4. [Next: Permissions] (次のステップ: 許可) を選択します。
5. `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` と `AWSElasticBeanstalkEnhancedHealth` の管理ポリシー、ならびにアプリケーションで必要なアクセス権限を付与するその他のポリシーがあればそれらもアタッチします。
6. [Next: Tags] (次へ: タグ) を選択します。
7. (オプション) ロールにタグを追加します。
8. [次へ: レビュー] を選択します。
9. ロールの名前を入力します。
10. [ロールの作成] を選択してください。

環境の作成時に、環境の作成ウィザードを使用するか、`eb create` コマンドで `--service-role` オプションを指定することで、カスタムサービスロールを適用します。

Elastic Beanstalk でのサービスにリンクされたロールの使用

AWS Elastic Beanstalk 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、Elastic Beanstalk 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 Elastic Beanstalk 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。

Elastic Beanstalk では、いくつかのタイプのサービスリンクロールが定義されています。

- モニタリングサービスにリンクされたロール - 実行中の環境の状態をモニタリングし、ヘルスイベント通知を発行することを、Elastic Beanstalk に許可します。
- メンテナンスサービスにリンクされたロール - 実行中の環境の定期的なメンテナンスアクティビティを実行することを、Elastic Beanstalk に許可します。
- マネージド更新サービスにリンクされたロール - 実行中の環境のスケジュールされたプラットフォーム更新を実行することを、Elastic Beanstalk に許可します。

トピック

- モニタリングサービスにリンクされたロール
- メンテナンスサービスにリンクされたロール
- マネージド更新サービスにリンクされたロール

モニタリングサービスにリンクされたロール

AWS Elastic Beanstalk 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、Elastic Beanstalk 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 Elastic Beanstalk 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ことで、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、Elastic Beanstalk の設定が簡単になります。Elastic Beanstalk には、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。特に定義しなければ、Elastic Beanstalk 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、不注意でリソースへのアクセスに必要なアクセス許可が削除されることがなくなり、Elastic Beanstalk 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 のサービスを参照して、[Service-Linked Role] (サービスにリンクされたロール) 列が[Yes] (はい) になっているサービスを探してください。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。

Elastic Beanstalk のサービスにリンクされたロールのアクセス許可

Elastic Beanstalk が AWSServiceRoleForElasticBeanstalk という名前のサービスにリンクされたロールを使用 - 実行中の環境の状態をモニタリングし、ヘルスイベント通知を発行することを、Elastic Beanstalk に許可します。

AWSServiceRoleForElasticBeanstalk サービスにリンクされたロールは、以下のサービスを信頼してロールを引き受けます。

- `elasticbeanstalk.amazonaws.com`

AWSServiceRoleForElasticBeanstalk 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、Elastic Beanstalk がお客様に代わってアクションを完了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。

AllowCloudformationReadOperationsOnElasticBeanstalkStacks

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudformationReadOperationsOnElasticBeanstalkStacks",
            "Effect": "Allow",
            "Action": [
                "cloudformation:DescribeStackResource",
                "cloudformation:DescribeStackResources",
                "cloudformation:DescribeStacks"
            ],
            "Resource": [
                "arn:aws:cloudformation:*:*:stack/awseb-*",
                "arn:aws:cloudformation:*:*:stack/eb-*"
            ]
        },
        {
            "Sid": "AllowOperations",
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeNotificationConfigurations",
                "autoscaling:DescribeScalingActivities",
                "autoscaling:PutNotificationConfiguration",
                "ec2:DescribeInstanceStatus",
                "ec2:AssociateAddress",
                "ec2:DescribeAddresses",
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "elasticloadbalancing:DescribeInstanceHealth",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetHealth",
                "elasticloadbalancing:DescribeTargetGroups",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sns:Publish"
            ],
            "Resource": [
```

```
                "*"
            ]
        }
    ]
}
```

サービスリンク役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) 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。

または、 AWS マネージドポリシーを使用して Elastic Beanstalk へのフルアクセスを提供することもできます。

Elastic Beanstalk 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。Elastic Beanstalk API を使用して Elastic Beanstalk 環境を作成するとき、サービスロールを指定しないと、Elastic Beanstalk 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。

> **Important**
>
> AWSServiceRoleForElasticBeanstalk サービスにリンクされたロールのサポートが開始された 2017 年 9 月 27 日より前に、Elastic Beanstalk サービスを使用していた場合、アカウントでそのロールが必要になると、Elastic Beanstalk によってアカウントに AWSServiceRoleForElasticBeanstalk ロールが作成されていました。詳細については、「IAM アカウントに新しいロールが表示される」を参照してください。

環境の作成時、Elastic Beanstalk が AWSServiceRoleForElasticBeanstalk サービスにリンクされたロールをアカウントに作成しようとすると、`iam:CreateServiceLinkedRole` アクセス許可が必要になります。このアクセス許可がない場合、環境の作成は失敗し、問題を説明するメッセージが表示されます。

別の方法として、サービスにリンクされたロールを作成するアクセス許可を持つ別のユーザーが IAM を使用して、サービスにリンクされたロールを事前に作成できます。この場合は、`iam:CreateServiceLinkedRole` アクセス許可がなくても、環境を作成できます。

お客様 (または別のユーザー) は、IAM コンソールを使用して、[Elastic Beanstalk] ユースケースでサービスにリンクされたロールを作成できます。IAM CLI または IAM API

で、`elasticbeanstalk.amazonaws.com` サービス名でサービスリンクロールを作成します。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの作成」を参照してください。このサービスリンクロールを削除しても、同じ方法でロールを再作成できます。

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。Elastic Beanstalk API を使用して Elastic Beanstalk 環境を作成するとき、サービスロールを指定しないと、Elastic Beanstalk 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。

Elastic Beanstalk のサービスにリンクされたロールの編集

Elastic Beanstalk では、AWSServiceRoleForElasticBeanstalk 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。

Elastic Beanstalk 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、積極的にモニタリングまたは保守さ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、サービスリンクロールのリソース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

サービスにリンクされたロールを IAM で削除する前に、すべての Elastic Beanstalk 環境が終了しているか、または別のサービスロールを使用していることを確認する必要があります。

> **Note**
>
> 環境を終了しようとしたときに、サービスにリンクされたロールを Elastic Beanstalk サービスが使用していると、終了は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。

AWSServiceRoleForElasticBeanstalk を使用する Elastic Beanstalk 環境を終了するには (コンソール)

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

EB CLI を使用した Elastic Beanstalk 環境の終了の詳細については、「eb terminate」を参照してください。

API を使用した Elastic Beanstalk 環境の終了の詳細については、「TerminateEnvironment」を参照してください。

サービスリンク役割の手動による削除

IAM コンソール、IAM CLI、または IAM API を使用して、AWSServiceRoleForElasticBeanstalk 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。

Elastic Beanstalk 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

Elastic Beanstalk は、サービスを利用できる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 Elastic Beanstalk エンドポイントとクォータ」を参照してください。

メンテナンスサービスにリンクされたロール

AWS Elastic Beanstalk 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、Elastic Beanstalk 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 Elastic Beanstalk 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ことで、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、Elastic Beanstalk の設定が簡単になります。Elastic Beanstalk には、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。特に定義しなければ、Elastic Beanstalk 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、不注意でリソースへのアクセスに必要なアクセス許可が削除されることがなくなり、Elastic Beanstalk 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 のサービスを参照して、[Service-Linked Role] (サービスにリンクされたロール) 列が[Yes] (はい) になっているサービスを探してください。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。

Elastic Beanstalk のサービスにリンクされたロールのアクセス許可

Elastic Beanstalk が AWSServiceRoleForElasticBeanstalkMaintenance という名前のサービスリンクロールを使用 - 実行中の環境の定期的なメンテナンスアクティビティを実行することを、Elastic Beanstalk に許可します。

AWSServiceRoleForElasticBeanstalkMaintenance サービスにリンクされたロールは、以下のサービスを信頼してロールを引き受けます。

- `maintenance.elasticbeanstalk.amazonaws.com`

AWSServiceRoleForElasticBeanstalkMaintenance 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、Elastic Beanstalk がお客様に代わってアクションを完了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。

JSON

```
{
    "Version": "2012-10-17",
    "Statement":
        {
            "Sid":
 "AllowCloudformationChangeSetOperationsOnElasticBeanstalkStacks",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateChangeSet",
                "cloudformation:DescribeChangeSet",
                "cloudformation:ExecuteChangeSet",
                "cloudformation:DeleteChangeSet",
                "cloudformation:ListChangeSets",
                "cloudformation:DescribeStacks"
            ],
            "Resource": [
                "arn:aws:cloudformation:*:*:stack/awseb-*",
                "arn:aws:cloudformation:*:*:stack/eb-*"
            ]
```

```
        }
}
```

サービスリンク役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) 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。

または、 AWS マネージドポリシーを使用して Elastic Beanstalk へのフルアクセスを提供することもできます。

Elastic Beanstalk 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。Elastic Beanstalk API を使用して Elastic Beanstalk 環境を作成するとき、インスタンスプロファイルを指定しないと、Elastic Beanstalk 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。

> **Important**
>
> このサービスリンク役割はこの役割でサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。AWSServiceRoleForElasticBeanstalkMaintenance サービスにリンクされたロールのサポートが開始された 2019 年 4 月 18 日より前に、Elastic Beanstalk サービスを使用していた場合、アカウントでそのロールが必要になると、Elastic Beanstalk によってアカウントに AWSServiceRoleForElasticBeanstalkMaintenance ロールが作成されていました。詳細については、「IAM アカウントに新しいロールが表示される」を参照してください。

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、同じ手順でアカウントにロールを再作成できます。Elastic Beanstalk API を使用して Elastic Beanstalk 環境を作成するとき、インスタンスプロファイルを指定しないと、Elastic Beanstalk 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。

Elastic Beanstalk のサービスにリンクされたロールの編集

Elastic Beanstalk では、AWSServiceRoleForElasticBeanstalkMaintenance 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。

Elastic Beanstalk 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、積極的にモニタリングまたは保守さ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、サービスリンクロールのリソース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ロールを削除する前に、このロールを使用しているすべての Elastic Beanstalk 環境を終了する必要があります。

> **Note**
>
> 環境を終了しようとしたときに、サービスにリンクされたロールを Elastic Beanstalk サービスが使用していると、終了は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。

AWSServiceRoleForElasticBeanstalkMaintenance を使用する Elastic Beanstalk 環境を終了するには (コンソール)

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション)、[Terminate environment] (環境の終了) の順に選択します。
4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

EB CLI を使用した Elastic Beanstalk 環境の終了の詳細については、「eb terminate」を参照してください。

API を使用した Elastic Beanstalk 環境の終了の詳細については、「TerminateEnvironment」を参照してください。

サービスリンク役割の手動による削除

IAM コンソール、IAM CLI、または IAM API を使用して、AWSServiceRoleForElasticBeanstalkMaintenance 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。

Elastic Beanstalk 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

Elastic Beanstalk は、サービスを利用できる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 Elastic Beanstalk エンドポイントとクォータ」を参照してください。

マネージド更新サービスにリンクされたロール

AWS Elastic Beanstalk 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、Elastic Beanstalk 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 Elastic Beanstalk 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ことで、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、Elastic Beanstalk の設定が簡単になります。Elastic Beanstalk には、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。特に定義しなければ、Elastic Beanstalk 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、不注意でリソースへのアクセスに必要なアクセス許可が削除されることがなくなり、Elastic Beanstalk 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 のサービスを参照して、[Service-Linked Role] (サービスにリンクされたロール) 列が[Yes] (はい) になっているサービスを探してください。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。

Elastic Beanstalk のサービスにリンクされたロールのアクセス許可

Elastic Beanstalk が AWSServiceRoleForElasticBeanstalkManagedUpdates という名前のサービスリンクロールを使用 - 実行中の環境のスケジュールされたプラットフォーム更新を実行することを、Elastic Beanstalk に許可します。

AWSServiceRoleForElasticBeanstalkManagedUpdates サービスにリンクされたロールは、以下のサービスを信頼してロールを引き受けます。

- `managedupdates.elasticbeanstalk.amazonaws.com`

AWSElasticBeanstalkManagedUpdatesServiceRolePolicy 管理ポリシーは、サービスにリンクされたロール AWSServiceRoleForElasticBeanstalkManagedUpdates に、Elastic Beanstalk がお客様に代わってマネージド更新アクションを完了するために必要なすべてのアクセス許可を付与します。管理ポリシーの内容を表示するには、「AWS マネージドポリシーリファレンスガイド」の「AWSElasticBeanstalkManagedUpdatesServiceRolePolicy」ページを参照してください。

サービスリンク役割の作成、編集、削除を IAM エンティティ (ユーザー、グループ、役割など) 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。

または、 AWS マネージドポリシーを使用して Elastic Beanstalk へのフルアクセスを提供することもできます。

Elastic Beanstalk 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。Elastic Beanstalk API を使用して Elastic Beanstalk 環境を作成するとき、マネージド更新を有効にし、`AWSServiceRoleForElasticBeanstalkManagedUpdates` 名前空間の `ServiceRoleForManagedUpdates` オプションの値として `aws:elasticbeanstalk:managedactions` を指定すると、Elastic Beanstalk 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。

環境を作成するときに Elastic Beanstalk がアカウントの AWSServiceRoleForElasticBeanstalkManagedUpdates サービスにリンクされたロールを作成しようとするときは、`iam:CreateServiceLinkedRole` アクセス許可が必要です。このアクセス許可がない場合、環境の作成は失敗し、問題を説明するメッセージが表示されます。

別の方法として、サービスにリンクされたロールを作成するアクセス許可を持つ別のユーザーが IAM を使用して、サービスにリンクされたロールを事前に作成できます。この場合は、`iam:CreateServiceLinkedRole` アクセス許可がなくても、環境を作成できます。

お客様 (または別のユーザー) は、IAM コンソールを使用して、[Elastic Beanstalk Managed Updates] ユースケースでサービスにリンクされたロールを作成できます。IAM CLI または IAM API で、`managedupdates.elasticbeanstalk.amazonaws.com` サービス名でサービスリンクロールを作成します。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの作成」を参照してください。このサービスリンクロールを削除しても、同じ方法でロールを再作成できます。

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。Elastic Beanstalk API を使用して Elastic Beanstalk 環境を作成すると

き、マネージド更新を有効にし、`AWSServiceRoleForElasticBeanstalkManagedUpdates` 名前空間の `ServiceRoleForManagedUpdates` オプションの値として `aws:elasticbeanstalk:managedactions` を指定すると、Elastic Beanstalk 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。

Elastic Beanstalk のサービスにリンクされたロールの編集

Elastic Beanstalk では、AWSServiceRoleForElasticBeanstalkManagedUpdates 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。

Elastic Beanstalk 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、積極的にモニタリングまたは保守さ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、サービスリンクロールのリソース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

サービスにリンクされたロールを IAM で削除する前に、マネージド更新が有効になっている Elastic Beanstalk 環境が終了しているか、または別のサービスロールを使用していることを確認する必要があります。

> **Note**
>
> 環境を終了しようとしたときに、サービスにリンクされたロールを Elastic Beanstalk サービスが使用していると、終了は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。

AWSServiceRoleForElasticBeanstalkManagedUpdates を使用する Elastic Beanstalk 環境を終了するには (コンソール)

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [アクション]、[環境の終了] の順に選択します。

4. 画面上のダイアログボックスを使用して、環境の終了を確認します。

EB CLI を使用した Elastic Beanstalk 環境の終了の詳細については、「eb terminate」を参照してください。

API を使用した Elastic Beanstalk 環境の終了の詳細については、「TerminateEnvironment」を参照してください。

サービスリンク役割の手動による削除

IAM コンソール、IAM CLI、または IAM API を使用して、AWSServiceRoleForElasticBeanstalkManagedUpdates 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。

Elastic Beanstalk 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

Elastic Beanstalk は、サービスを利用できる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 Elastic Beanstalk エンドポイントとクォータ」を参照してください。

Elastic Beanstalk ユーザーポリシーの管理

AWS Elastic Beanstalk には、Elastic Beanstalk が管理するすべてのリソースへのフルアクセスまたは読み取り専用アクセスを割り当てることができる 2 つの管理ポリシーが用意されています。ポリシーは、 AWS Identity and Access Management (IAM) ユーザーまたはグループ、またはユーザーが引き受けるロールにアタッチできます。

管理ユーザーポリシー

- AdministratorAccess-AWSElasticBeanstalk – Elastic Beanstalk アプリケーション、アプリケーションバージョン、設定の構成、環境、およびそれらの基盤となるリソースを作成、変更、削除するための、完全な管理者権限をユーザーに付与します。マネージドポリシーの内容を表示するには、「AWS マネージドポリシーリファレンスガイド」の「AdministratorAccess-AWSElasticBeanstalk」ページを参照してください。
- AWSElasticBeanstalkReadOnly – ユーザーにアプリケーションおよび環境を表示することを許可します。ただし、それらを変更するオペレーションは許可しません。すべての Elastic Beanstalk リソース、および Elastic Beanstalk コンソールが取得する他の AWS リソースへの読み取り専用アクセスを提供します。読み取り専用アクセスでは、Elastic Beanstalk ログを読むためのダウン

ロードなどのアクションは実行できません。これは、Amazon S3 バケットにログがステージングされるので、Elastic Beanstalk にそのバケットに対する書き込みアクセス許可が必要なためです。Elastic Beanstalk ログへのアクセスを有効にする方法については、このトピックの最後の例を参照してください。マネージドポリシーの内容を表示するには、「AWS マネージドポリシーリファレンスガイド」の「AWSElasticBeanstalkReadOnly」ページを参照してください。

> **Important**
>
> Elastic Beanstalk マネージド型ポリシーは、詳細なアクセス権限を提供しません。Elastic Beanstalk アプリケーションの操作に必要となる可能性のある、すべてのアクセス権限が付与されます。場合によっては、管理ポリシーのアクセス許可をさらに制限することもできます。1 つのユースケースの例については、「環境間の Amazon S3 バケットアクセスの防止」を参照してください。
> また、当社の管理ポリシーでは、Elastic Beanstalk では管理されておらず、お客様によりソリューションに追加されるような、カスタムリソースのためのアクセス許可についてもサポートしていません。よりきめの細かなアクセス許可、必要最小限のアクセス許可、またはリソースに対するアクセス許可をカスタムで作成するには、カスタムポリシーを使用します。

非推奨の マネージドポリシー

以前は、Elastic Beanstalk では、AWSElasticBeanstalkFullAccess と AWSElasticBeanstalkReadOnlyAccess という、他の 2 つの管理ユーザーポリシーをサポートしていました。これらの古いポリシーは廃止する予定です。これらは、IAM コンソールで、引き続き表示および使用できる場合があります。ただし、新しい管理ユーザーポリシーを使用することと、(それが存在する場合は) カスタムポリシーを追加して、カスタムリソースにアクセス許可を付与することをお勧めします。

他のサービスとの統合に関するポリシー

また、必要に応じて、環境を他のサービスと統合することを許可する、より詳細なポリシーも指定します。

- AWSElasticBeanstalkRoleCWL – 環境が Amazon CloudWatch Logs ロググループを管理できるようにします。
- AWSElasticBeanstalkRoleRDS – 環境が Amazon RDS インスタンスを統合できるようにします。

- AWSElasticBeanstalkRoleWorkerTier – ワーカー環境階層が Amazon DynamoDB テーブルと Amazon SQS キューを作成できるようにします。
- AWSElasticBeanstalkRoleECS – マルチコンテナ Docker 環境が Amazon ECS クラスターを管理できるようにします。
- AWSElasticBeanstalkRoleCore – ウェブサービス環境のコアオペレーションができるようにします。
- AWSElasticBeanstalkRoleSNS – 環境が Amazon SNS トピック統合を有効にできるようにします。

特定の管理ポリシーの JSON ソースを確認するには、「AWS 管理ポリシーリファレンスガイド」を参照してください。

管理ポリシーによるアクセスのコントロール

管理ポリシーにより、Elastic Beanstalk へのフルアクセスまたは読み取り専用アクセスを許可できます。Elastic Beanstalk は、新しい機能へのアクセスに追加の許可が必要になった場合に、これらのポリシーを自動的に更新します。

管理ポリシーを IAM ユーザーまたはグループに適用するには

1. IAM コンソールの [Policies (ポリシー)] ページを開きます。
2. 検索ボックスに「**AWSElasticBeanstalk**」と入力して、ポリシーを絞り込みます。
3. ポリシーのリストで、[AWSElasticBeanstalkReadOnly] または [AdministratorAccess-AWSElasticBeanstalk] の横にあるチェックボックスをオンにします。
4. [Policy actions] を選択して、[Attach] を選択します。
5. ポリシーをアタッチするユーザーとグループを 1 つ以上選択します。[Filter] メニューと検索ボックスを使用して、プリンシパルエンティティのリストをフィルタリングできます。
6. Attach policy] (ポリシーのアタッチ) を選択してください。

カスタムユーザーポリシーの作成

独自の IAM ポリシーを作成して、特定の Elastic Beanstalk リソースに対する特定の Elastic Beanstalk API アクションを許可または拒否したり、Elastic Beanstalk で管理されていないカスタムリソースへのアクセスを制御したりできます。ユーザーまたはグループへのポリシーのアタッチについては、「IAM ユーザーガイド」の「ポリシーの使用」を参照してください。カスタムポリシーの作成の詳細については、「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。

> **Note**
>
> なお、現時点では、ユーザーによる Elastic Beanstalk API の使用方法は制限できますが、基盤となるリソースを作成するアクセス許可を持つユーザーが Amazon EC2 といったサービスで他のリソースを作成することを防ぐ効果的な方法はありません。
> これらのポリシーを、すべての基盤となるリソースを確保する手段としてではなく、Elastic Beanstalk を配布する効果的な方法と考えてください。

> **Important**
>
> Elastic Beanstalk サービスロールにカスタムポリシーが割り当てられている場合は、起動テンプレートの適切なアクセス許可を割り当てることが重要です。それ以外の場合は、環境を更新したり、新しい環境を起動したりするために必要なアクセス許可がない可能性があります。詳細については、「起動テンプレートに必要なアクセス許可」を参照してください。

IAM ポリシーには、付与するアクセス許可を定義したポリシーステートメントが含まれています。Elastic Beanstalk のポリシーステートメントを作成する場合、ポリシーステートメントを構成する以下の 4 つの部分について、使い方を把握しておく必要があります。

- 効果は、ステートメントのアクションを許可または拒否するかどうかを指定します。
- アクションは、制御する API オペレーションを指定します。たとえば、`elasticbeanstalk:CreateEnvironment` を使用して `CreateEnvironment` オペレーションを指定します。環境の作成といった特定のオペレーションには、これらのアクションを実行するための追加の許可が必要となります。詳細については、「Elastic Beanstalk アクションのリソースと条件」を参照してください。

  > **Note**
  >
  > `UpdateTagsForResource` API オペレーションを使用するには、API オペレーション名ではなく、2 つの仮想アクションのいずれか (または両方) を指定します。
  >
  > `elasticbeanstalk:AddTags`
  >
  > `UpdateTagsForResource` を呼び出し、追加するタグのリストを `TagsToAdd` パラメータで渡すアクセス許可を制御します。

elasticbeanstalk:RemoveTags

UpdateTagsForResource を呼び出し、削除するタグキーのリストを TagsToRemove パラメータで渡すアクセス許可を制御します。

- リソースは、アクセスをコントロールする対象のリソースを指定します。Elastic Beanstalk リソースを指定するには、リソースの Amazon リソースネーム (ARN) のリストを指定します。
- (オプション) 条件は、ステートメントで付与された許可に対する制限を指定します。詳細については、「Elastic Beanstalk アクションのリソースと条件」を参照してください。

以下のセクションでは、カスタムユーザーポリシーを考慮する可能性のあるいくつかのケースを示します。

制限付き Elastic Beanstalk 環境の作成の有効化

次のサンプルポリシーでは、このポリシーを添付されたユーザーは、CreateEnvironment アクションを呼び出し、指定されたアプリケーションとアプリケーションバージョンを使用して、名前が **Test** で始まる環境を作成できます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid":"CreateEnvironmentPerm",
      "Action": [
        "elasticbeanstalk:CreateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My First
 Elastic Beanstalk Application/Test*"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My First Elastic Beanstalk Application"],
```

```
          "elasticbeanstalk:FromApplicationVersion":
 ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My First
 Elastic Beanstalk Application/First Release"]
        }
      }
    },
    {
      "Sid":"AllNonResourceCalls",
      "Action":[
        "elasticbeanstalk:CheckDNSAvailability",
        "elasticbeanstalk:CreateStorageLocation"
      ],
      "Effect":"Allow",
      "Resource":[
        "*"
      ]
    }
  ]
}
```

上記のポリシーは、Elastic Beanstalk オペレーションへの制限付きアクセスを許可する方法を示しています。環境を実際に起動するには、環境を強化する AWS リソースを作成するアクセス許可がユーザーに必要です。たとえば、次のポリシーはウェブサーバー環境の一連のデフォルトリソースへのアクセス権を付与します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:*",
        "ecs:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
```

```
        "sqs:*"
        ],
      "Resource": "*"
    }
  ]
}
```

Amazon S3 に保存されている Elastic Beanstalk ログへのアクセスの有効化

以下の例のポリシーでは、ユーザーは Elastic Beanstalk ログをプルし、Amazon S3 にステージングして、取得できます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:DeleteObject",
        "s3:GetObjectAcl",
        "s3:PutObjectAcl"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::elasticbeanstalk-*"
    }
  ]
}
```

> **Note**
>
> これらのアクセス許可をログのパスのみに制限するには、次のリソース形式を使用します。
>
> ```
> "arn:aws:s3:::elasticbeanstalk-us-east-2-123456789012/resources/environments/logs/*"
> ```

特定の Elastic Beanstalk アプリケーションの管理の有効化

以下のサンプルポリシーでは、ユーザーが 1 つの特定の Elastic Beanstalk アプリケーション内の環境およびその他のリソースを管理できるようにします。このポリシーは、他のアプリケーションのリソースに対する Elastic Beanstalk アクションを拒否し、Elastic Beanstalk アプリケーションの作成と削除も拒否します。

> **Note**
>
> このポリシーは、他のサービスを介したリソースへのアクセスを拒否しません。基盤となるリソースを確保する方法としてではなく、さまざまなユーザー間で Elastic Beanstalk アプリケーションを管理するための責任を分散する効果的な方法を示しています。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk:DeleteApplication"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:CreateApplicationVersion",
        "elasticbeanstalk:CreateConfigurationTemplate",
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk:DeleteApplicationVersion",
        "elasticbeanstalk:DeleteConfigurationTemplate",
        "elasticbeanstalk:DeleteEnvironmentConfiguration",
        "elasticbeanstalk:DescribeApplicationVersions",
        "elasticbeanstalk:DescribeConfigurationOptions",
        "elasticbeanstalk:DescribeConfigurationSettings",
```

```
        "elasticbeanstalk:DescribeEnvironmentResources",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticbeanstalk:DescribeEvents",
        "elasticbeanstalk:DeleteEnvironmentConfiguration",
        "elasticbeanstalk:RebuildEnvironment",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RestartAppServer",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "elasticbeanstalk:SwapEnvironmentCNAMEs",
        "elasticbeanstalk:TerminateEnvironment",
        "elasticbeanstalk:UpdateApplicationVersion",
        "elasticbeanstalk:UpdateConfigurationTemplate",
        "elasticbeanstalk:UpdateEnvironment",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "elasticbeanstalk:ValidateConfigurationSettings"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "elasticbeanstalk:InApplication": [
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
myapplication"
          ]
        }
      }
    }
  ]
}
```

Elastic Beanstalk の Amazon リソースネームの形式

リソースの Amazon リソースネーム (ARN) を使用して、IAM ポリシーのリソースを指定します。Elastic Beanstalk では、ARN の形式は以下のとおりです。

```
arn:aws:elasticbeanstalk:region:account-id:resource-type/resource-path
```

コードの説明は以下のとおりです。

- *region* は、リソースが存在するリージョンです (**us-west-2** など)。

- *account-id* は AWS アカウント ID で、ハイフンはありません (例: **123456789012**)
- *resource-type* は、Elastic Beanstalk リソースのタイプ (など) を識別します。environmentすべての Elastic Beanstalk リソースタイプのリストについては、以下の表を参照してください。
- *resource-path* は、特定のリソースを識別する部分です。Elastic Beanstalk リソースには、そのリソースを一意に識別するパスがあります。各リソースタイプのリソースパスの形式については、以下の表を参照してください。例えば、アプリケーションには必ず環境が関連付けられています。アプリケーション **myApp** の環境 **myEnvironment** のリソースパスは次のようになります。

```
myApp/myEnvironment
```

Elastic Beanstalk には複数のリソースタイプがあり、ポリシーで指定できます。次の表は、各リソースタイプの ARN 形式と例を示しています。

リソースタイプ	ARN フォーマット
application	arn:aws:elasticbeanstalk: *region*:*account-id* :application/ *application-name* 例: **arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App**
applicationversion	arn:aws:elasticbeanstalk: *region*:*account-id* :applicationversion/ *application-name* /*version-label* 例: **arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version**
configurationtemplate	arn:aws:elasticbeanstalk: *region*:*account-id* :configurationtemplate/ *application-name* /*template-name* 例: **arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template**
environment	arn:aws:elasticbeanstalk: *region*:*account-id* :environment/ *application-name* /*environment-name*

リソースタイプ	ARN フォーマット
	例: **arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/MyEnvironment**
platform	arn:aws:elasticbeanstalk: *region*:*account-id* :platform / *platform-name* /*platform-version* 例: **arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/MyPlatform/1.0**
solutionstack	arn:aws:elasticbeanstalk: *region*::solutionstack/ *solutionstack-name* 例: **arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7**

特定のアプリケーションには必ず環境、アプリケーションバージョン、および設定テンプレートが含まれます。このすべてのリソースのパスにアプリケーション名が含まれています。したがって、リソースは、名前と自身が属するアプリケーションによって一意に識別されます。ソリューションスタックは設定テンプレートと環境で使用されますが、ソリューションスタックはアプリケーションや AWS アカウントに固有ではなく、ARN ARNs にアプリケーションや AWS アカウントがありません。

Elastic Beanstalk アクションのリソースと条件

このセクションでは、ポリシーステートメントで使用できるリソースと条件について説明します。このポリシーステートメントは、特定の Elastic Beanstalk リソースに対する特定の Elastic Beanstalk アクションの実行を許可します。

条件を使用すると、アクションが完了する必要があるリソースに対する許可を指定できます。例えば、`CreateEnvironment` アクションを呼び出すときは、デプロイするアプリケーションバージョンと、そのアプリケーション名が含まれるアプリケーションも指定する必要があります。`CreateEnvironment` アクションの許可を設定するとき、`InApplication` および `FromApplicationVersion` 条件を使用して、アクションの実行対象アプリケーションとアプリケーションバージョンを指定します。

さらに、ソリューションスタック (FromSolutionStack) または設定テンプレート (FromConfigurationTemplate) で環境設定を指定することもできます。次のポリシーステートメントでは、CreateEnvironment アクションが、アプリケーション **My App** (InApplication 条件で指定) で、**myenv** (Resource で指定) という名前の環境を、アプリケーションバージョン **My Version** (FromApplicationVersion) と **32bit Amazon Linux running Tomcat 7** 設定 (FromSolutionStack) を使用して作成できるようにします。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/
myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromApplicationVersion":
 ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My
 Version"],
          "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-
east-2::solutionstack/32bit Amazon Linux running Tomcat 7"]
        }
      }
    }
  ]
}
```

> **Note**
>
> このトピックで説明されているほとんどの条件キーは Elastic Beanstalk に固有のものであり、その名前には elasticbeanstalk: プレフィックスが含まれています。簡潔にす

るために、以降のセクションで言及するときは、条件キー名からこのプレフィックスを省略します。たとえば、フルネームの elasticbeanstalk:InApplication の代わりに InApplication を使用します。
対照的に、 AWS サービス間で使用されるいくつかの条件キーについて言及し、そのaws:プレフィックスを含めて例外を強調します。
ポリシーの例には、常にプレフィックスを含む完全条件キーの名前が表示されます。

セクション

- Elastic Beanstalk アクションのポリシーの情報
- Elastic Beanstalk アクションの条件キー

Elastic Beanstalk アクションのポリシーの情報

以下の表は、すべての Elastic Beanstalk アクション、各アクションの対象リソース、および条件によって提供できる追加コンテキスト情報のリストです。

Elastic Beanstalk アクションのポリシーの情報 (リソース、条件、例、依存関係など)

リソース	条件	ステートメント例
アクション: AbortEnvironmentUpdate		
application environment	aws:ResourceTag/ *key-name* (オプション) aws:TagKeys (オプション)	次のポリシーでは、アプリケーション My App の環境に対する更新オペレーションを中止することをユーザーに許可します。 JSON { "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:AbortEnvironmentUpdate"], "Effect": "Allow", "Resource": [

リソース	条件	ステートメント例

		JSON
		`"arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"`
		`]`
		`}`
		`]`
		`}`

アクション: CheckDNSAvailability		

"*"	該当なし	JSON
		`{`
		`"Version": "2012-10-17",`
		`"Statement": [`
		`{`
		`"Action": [`
		`"elasticbeanstalk:CheckDNSAvailability"`
		`],`
		`"Effect": "Allow",`
		`"Resource": "*"`
		`}`
		`]`
		`}`

アクション: ComposeEnvironments		

リソース	条件	ステートメント例

application	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、アプリケーション My App に属する環境を構成することをユーザーに許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:ComposeEnvironments"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App"
		]
		}
		]
		}

アクション: CreateApplication		

リソース	条件	ステートメント例

application	aws:RequestTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		この例では、CreateApplication アクションに対して、**DivA** で始まる名前のアプリケーションを作成することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:CreateApplication"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:application/DivA*"
		]
		}
		]
		}

アクション: CreateApplicationVersion		

リソース	条件	ステートメント例

applicationversion	InApplication
	
	aws:RequestTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		この例では、CreateApplicationVersion アクションに対して、アプリケーション **My App** で任意の名前 (*) を持つアプリケーションバージョンを作成することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:CreateApplicationVersion"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/*"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

アクション: CreateConfigurationTemplate

リソース	条件	ステートメント例

configurationtemplate	InApplication
	
	FromApplication
	
	FromApplicationVersion
	
	FromConfigurationTemplate
	
	FromEnvironment
	
	FromSolutionStack
	
	aws:RequestTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、CreateConfigurationTemplate アクションに対して、アプリケーション **My App** で、名前が **My Template** (My Template*) で始まる設定テンプレートを作成することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:CreateConfigurationTemplate"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template*"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"],
		"elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7"]
		}
		}
		}
		]
		}

リソース	条件	ステートメント例
アクション: `CreateEnvironment`		

リソース	条件	ステートメント例

environment	InApplication
	
	FromApplicationVersion
	
	FromConfigurationTemplate
	
	FromSolutionStack
	
	aws:RequestTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		以下のポリシーでは、CreateEnvironment アクションが、アプリケーション **My App** で **myenv** という名前の環境を、ソリューションスタック **32bit Amazon Linux running Tomcat 7** を使用して作成できるようにします。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:CreateEnvironment"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"],
		"elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"],
		"elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7"]
		}
		}
		}

リソース	条件	ステートメント例

		``
		  ]
		}
		``

アクション: CreatePlatformVersion		

platform	aws:RequestTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		この例では、CreatePlatformVersion アクションに対して、名前が **us-east-2_** で始まる、us-east-2 リージョンを対象とするプラットフォームバージョンを作成することを許可します。
		
		JSON
		
		``
		{
		  "Version": "2012-10-17",
		  "Statement": [
		    {
		      "Action": [
		        "elasticbeanstalk:CreatePlatformVersion"
		      ],
		      "Effect": "Allow",
		      "Resource": [
		        "arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*"
		      ]
		    }
		  ]
		}
		``

アクション: CreateStorageLocation		

リソース	条件	ステートメント例

"*"	該当なし	JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:CreateStorageLocation"
		],
		"Effect": "Allow",
		"Resource": "*"
		}
		]
		}

アクション: DeleteApplication		

リソース	条件	ステートメント例

application	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、DeleteApplication アクションに対して、アプリケーション **My App** を削除することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:DeleteApplication"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"
		]
		}
		]
		}

アクション: DeleteApplicationVersion		

リソース	条件	ステートメント例

applicationversion	InApplication
	
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、DeleteApplicationVersion アクションに対して、アプリケーション **My App** で **My Version** という名前のアプリケーションバージョンを削除することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:DeleteApplicationVersion"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

アクション: DeleteConfigurationTemplate

リソース	条件	ステートメント例
configurationtemplate	InApplication (オプション) aws:ResourceTag/ *key-name* (オプション) aws:TagKeys (オプション)	以下のポリシーでは、DeleteConfigurationTemplate アクションが、アプリケーション **My App** で **My Template** という名前の設定テンプレートを削除できるようにします。アプリケーション名を条件として指定するかどうかはオプションです。 JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DeleteConfigurationTemplate" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template" ] } ] }`
アクション: DeleteEnvironmentConfiguration		

リソース	条件	ステートメント例
environment	InApplication (オプション)	以下のポリシーでは、DeleteEnvironmentConfiguration アクションが、アプリケーション **My App** で環境 **myenv** のドラフト設定を削除できるようにします。アプリケーション名を条件として指定するかどうかはオプションです。 JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DeleteEnvironmentConfiguration" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv" ] } ] }`
アクション: DeletePlatformVersion		

リソース	条件	ステートメント例

platform	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		以下のポリシーでは、DeletePlatformVersion アクションに対して、名前が **us-east-2_** で始まる、us-east-2 リージョンを対象とするプラットフォームバージョンを削除することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:DeletePlatformVersion"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*"
		]
		}
		]
		}

アクション: DescribeApplications		

リソース	条件	ステートメント例

application	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、DescribeApplications アクションに対して、アプリケーション My App を説明することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:DescribeApplications"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"
		]
		}
		]
		}

アクション: DescribeApplicationVersions		

リソース	条件	ステートメント例
applicationversion	InApplication (オプション) aws:ResourceTag/ *key-name* (オプション) aws:TagKeys (オプション)	以下のポリシーでは、DescribeApplicationVersions アクションが、アプリケーション **My App** でアプリケーションバージョン **My Version** の定義を表示できるようにします。アプリケーション名を条件として指定するかどうかはオプションです。 JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DescribeApplicationVersions" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version" ] } ] }`
アクション: DescribeConfigurationOptions		

リソース	条件	ステートメント例
environment configurationtemplate solutionstack	InApplication (オプション) aws:ResourceTag/ *key-name* (オプション) aws:TagKeys (オプション)	以下のポリシーでは、DescribeConfigurationOptions アクションが、アプリケーション **My App** で環境 **myenv** の設定オプションの定義を表示できるようにします。アプリケーション名を条件として指定するかどうかはオプションです。 JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": "elasticbeanstalk:DescribeConfigurationOptions", "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv" ] } ] }`
アクション: DescribeConfigurationSettings		

リソース	条件	ステートメント例
environment configurationtemplate	InApplication (オプション) aws:ResourceTag/ *key-name* (オプション) aws:TagKeys (オプション)	以下のポリシーでは、DescribeConfigurationSettings アクションが、アプリケーション **My App** で環境 **myenv** の設定の定義を表示できるようにします。アプリケーション名を条件として指定するかどうかはオプションです。 JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": "elasticbeanstalk:DescribeConfigurationSettings", "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv" ] } ] }`
アクション: DescribeEnvironmentHealth		

リソース	条件	ステートメント例

environment	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、DescribeEnvironmentHealth を使用して環境 **myenv** のヘルス情報を取得することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": "elasticbeanstalk:DescribeEnvironmentHealth",
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
		]
		}
		]
		}

アクション: DescribeEnvironmentResources		

リソース	条件	ステートメント例
environment	InApplication (オプション) aws:ResourceTag/ *key-name* (オプション) aws:TagKeys (オプション)	次のポリシーでは、 DescribeEnvironmentResources アクションがアプリケーション **myenv**の環境 の AWS リソースのリストを返すことを許可します**My App**。アプリケーション名を条件として指定するかどうかはオプションです。 JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": "elasticbeanstalk:DescribeEnvironmentResources", "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv" ] } ] }`
アクション: DescribeEnvironments		

リソース	条件	ステートメント例
environment	InApplication (オプション) aws:ResourceTag/ *key-name* (オプション) aws:TagKeys (オプション)	以下のポリシーでは、DescribeEnvironments アクションが、アプリケーション **My App** で環境 **myenv** と **myotherenv** の定義を表示できるようにします。アプリケーション名を条件として指定するかどうかはオプションです。 JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": "elasticbeanstalk:DescribeEnvironments", "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv", "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App2/myotherenv" ] } ] }`
アクション: DescribeEvents		

リソース	条件	ステートメント例
application applicationversion configurationtemplate environment	InApplication aws:ResourceTag/ *key-name* (オプション) aws:TagKeys (オプション)	次のポリシーでは、DescribeEvents アクションに対して、アプリケーション **My App** の環境 **myenv** とアプリケーションバージョン **My Version** に関するイベントの説明を一覧表示することを許可します。 JSON `{ "Version": "2012-10-17", "Statement": [{ "Action": "elasticbeanstalk:DescribeEvents", "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv", "arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }`

アクション: DescribeInstancesHealth

リソース	条件	ステートメント例

environment	該当なし	以下のポリシーでは、DescribeInstancesHealth を使用して **myenv** という名前の環境内のインスタンスのヘルス情報を取得できるようにします。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": "elasticbeanstalk:DescribeInstancesHealth",
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
		]
		}
		]
		}

アクション: DescribePlatformVersion		

リソース	条件	ステートメント例

platform	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		以下のポリシーでは、DescribePlatformVersion アクションに対して、名前が **us-east-2_** で始まる、us-east-2 リージョンを対象とするプラットフォームバージョンを説明することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:DescribePlatformVersion"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*"
		]
		}
		]
		}

アクション: ListAvailableSolutionStacks		

リソース	条件	ステートメント例

solutionstack	該当なし	次のポリシーでは、ListAvailableSolutionStacks アクションに対して、ソリューションスタック **32bit Amazon Linux running Tomcat 7** のみを返すことを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:ListAvailableSolutionStacks"
		],
		"Effect": "Allow",
		"Resource": "arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7"
		}
		]
		}

アクション: ListPlatformVersions

リソース	条件	ステートメント例

platform	aws:RequestTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		この例では、CreatePlatformVersion アクションに対して、名前が **us-east-2_** で始まる、us-east-2 リージョンを対象とするプラットフォームバージョンを作成することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:ListPlatformVersions"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*"
		]
		}
		]
		}

アクション: ListTagsForResource		

リソース	条件	ステートメント例

application
 	
applicationversion
 	
configurationtemplate
 	
environment
 	
platform
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、値が test のタグ stage がある場合にのみ、既存のリソースのタグを一覧表示することを ListTagsForResource アクションに許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:ListTagsForResource"
		],
		"Effect": "Allow",
		"Resource": "*",
		"Condition": {
		"StringEquals": {
		"aws:ResourceTag/stage": ["test"]
		}
		}
		}
		]
		}

アクション: RebuildEnvironment		

リソース	条件	ステートメント例

environment	InApplication
	
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、RebuildEnvironment アクションに対して、アプリケーション **My App** の環境 **myenv** を再構築することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:RebuildEnvironment"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

アクション: RequestEnvironmentInfo		

リソース	条件	ステートメント例

environment	InApplication
	
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、RequestEnvironmentInfo アクションに対して、アプリケーション **My App** の環境 **myenv** に関する情報をコンパイルすることを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:RequestEnvironmentInfo"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

アクション: RestartAppServer		

リソース	条件	ステートメント例

environment	InApplication	以下のポリシーでは、RestartAppServer アクションが、アプリケーション **My App** で環境 **myenv** のアプリケーションコンテナサーバーを再起動できるようにします。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:RestartAppServer"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

アクション: RetrieveEnvironmentInfo		

リソース	条件	ステートメント例

environment	InApplication
	
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、RetrieveEnvironmentInfo アクションに対して、アプリケーション **My App** の環境 **myenv** に関するコンパイル済み情報を取得することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:RetrieveEnvironmentInfo"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

アクション: SwapEnvironmentCNAMEs		

リソース	条件	ステートメント例

environment	InApplication (オプション)
	
	FromEnvironment (オプション)
		以下のポリシーでは、SwapEnvironmentCNAMEs アクションが、環境 **mysrcenv** と **mydestenv** の CNAME を交換できるようにします。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:SwapEnvironmentCNAMEs"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/MyApp/mysrcenv",
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/MyApp/mydestenv"
		]
		}
		]
		}

アクション: TerminateEnvironment		

リソース	条件	ステートメント例

environment	InApplication
	
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		以下のポリシーでは、TerminateEnvironment アクションが、アプリケーション **My App** で環境 **myenv** を削除できるようにします。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:TerminateEnvironment"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

アクション: UpdateApplication		

リソース	条件	ステートメント例

application	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、UpdateApplication アクションに対して、アプリケーション **My App** のプロパティを更新することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:UpdateApplication"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"
		]
		}
		]
		}

アクション: UpdateApplicationResourceLifecycle		

リソース	条件	ステートメント例

application	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、 UpdateApplicationResourceLifecycle アクションに対して、アプリケーション **My App** のライフサイクル設定を更新することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:UpdateApplicationResourceLifecycle"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"
		]
		}
		]
		}

アクション: UpdateApplicationVersion		

リソース	条件	ステートメント例

applicationversion	InApplication
	
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、UpdateApplicationVersion アクションに対して、アプリケーション **My App** のアプリケーションバージョン **My Version** のプロパティを更新することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:UpdateApplicationVersion"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

アクション: UpdateConfigurationTemplate

リソース	条件	ステートメント例

configurationtemplate	InApplication
	
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、UpdateConfigurationTemplate アクションに対して、アプリケーション **My App** で設定テンプレート **My Template** のプロパティまたはオプションを更新することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:UpdateConfigurationTemplate"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

アクション: UpdateEnvironment		

リソース	条件	ステートメント例
environment	InApplication FromApplicationVersion FromConfigurationTemplate aws:ResourceTag/ *key-name* (オプション) aws:TagKeys (オプション)	以下のポリシーでは、アプリケーションバージョン **My Version** をデプロイすることにより、UpdateEnvironment アクションが、アプリケーション **My App** で環境 **myenv** を更新できるようにします。 JSON `{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:UpdateEnvironment"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"], "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"] } } }] }`

リソース	条件	ステートメント例
アクション:UpdateTagsForResource – AddTags		

application
 	
applicationversion
 	
configurationtemplate
 	
environment
 	
platform
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:RequestTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		AddTags アクションは、UpdateTagsForResource API に関連付けられた 2 つの仮想アクションのうちの 1 つです。
		
		次のポリシーでは、値が test のタグ stage がある場合にのみ、既存のリソースのタグを変更することを AddTags アクションに許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:AddTags"
		],
		"Effect": "Allow",
		"Resource": "*",
		"Condition": {
		"StringEquals": {
		"aws:ResourceTag/stage": ["test"]
		}
		}
		}
		]
		}

アクション:UpdateTagsForResource – RemoveTags		

リソース	条件	ステートメント例

application
 	
applicationversion
 	
configurationtemplate
 	
environment
 	
platform
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		RemoveTags アクションは、UpdateTagsForResource API に関連付けられた 2 つの仮想アクションのうちの 1 つです。
		
		次のポリシーでは、既存の環境からタグ stage の削除をリクエストすることを RemoveTags アクションに拒否します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:RemoveTags"
		],
		"Effect": "Deny",
		"Resource": "*",
		"Condition": {
		"ForAnyValue:StringEquals": {
		"aws:TagKeys": ["stage"]
		}
		}
		}
		]
		}

アクション: ValidateConfigurationSettings		

リソース	条件	ステートメント例

template
 	
environment
	InApplication
	
	aws:ResourceTag/ *key-name* (オプション)
	
	aws:TagKeys (オプション)
		次のポリシーでは、ValidateConfigurationSettings アクションに対して、アプリケーション **My App** の環境 **myenv** の設定を検証することを許可します。
		
		JSON
		
		{
		"Version": "2012-10-17",
		"Statement": [
		{
		"Action": [
		"elasticbeanstalk:ValidateConfigurationSettings"
		],
		"Effect": "Allow",
		"Resource": [
		"arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
		],
		"Condition": {
		"StringEquals": {
		"elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
		}
		}
		}
		]
		}

Elastic Beanstalk アクションの条件キー

キーを使用すると、依存関係を表す条件を指定したり、許可を制限したり、アクションの入力パラメータに対する制約を指定したりできます。Elastic Beanstalk は、以下のキーをサポートしています。

`InApplication`

アクションの実行対象のリソースが含まれるアプリケーションを指定します。

次の例では、`UpdateApplicationVersion` アクションが、アプリケーションバージョン **My Version** のプロパティを更新できるようにします。`InApplication` 条件は、**My App** を **My Version** のコンテナとして指定します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateApplicationVersion"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My
 App/My Version"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

`FromApplicationVersion`

アプリケーションバージョンを、入力パラメータで依存関係または制約として指定します。

次の例では、UpdateEnvironment アクションが、アプリケーション **My App** で環境 **myenv** を更新できるようにします。FromApplicationVersion 条件は VersionLabel パラメータを制限して、アプリケーションバージョン **My Version** しか環境を更新できないようにします。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/
myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromApplicationVersion":
 ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/
My Version"]
        }
      }
    }
  ]
}
```

FromConfigurationTemplate

設定テンプレートを、入力パラメータで依存関係または制約として指定します。

次の例では、UpdateEnvironment アクションが、アプリケーション **My App** で環境 **myenv** を更新できるようにします。FromConfigurationTemplate 条件は TemplateName パラメータを制限して、設定テンプレート **My Template** しか環境を更新できないようにします。

JSON

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

```
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/
myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromConfigurationTemplate":
 ["arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My
 App/My Template"]
        }
      }
    }
  ]
}
```

`FromEnvironment`

環境を、入力パラメータで依存関係または制約として指定します。

次の例を使用すると、名前が **mysrcenv** および **mydestenv** で始まるすべての環境の **My App** で、`SwapEnvironmentCNAMEs` アクションが CNAME を交換できます。ただし、名前が **mysrcenvPROD*** と **mydestenvPROD*** で始まる環境は除きます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:SwapEnvironmentCNAMEs"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/
mysrcenv*",
```

```
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/
mydestenv*"
      ],
      "Condition": {
        "StringNotLike": {
          "elasticbeanstalk:FromEnvironment": [
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My
 App/mysrcenvPROD*",
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My
 App/mydestenvPROD*"
          ]
        }
      }
    }
  ]
}
```

FromSolutionStack

ソリューションスタックを、入力パラメータで依存関係または制約として指定します。

以下のポリシーでは、CreateConfigurationTemplate アクションが、アプリケーション **My App** で、名前が **My Template** (My Template*) で始まる設定テンプレートを作成できるようにします。FromSolutionStack 条件は、ソリューションスタック **32bit Amazon Linux running Tomcat 7** のみをそのパラメータの入力値として許可するように solutionstack パラメータを制限します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateConfigurationTemplate"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-
east-2:123456789012:configurationtemplate/My App/My Template*"
      ],
      "Condition": {
        "StringEquals": {
```

```
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-
east-2::solutionstack/32bit Amazon Linux running Tomcat 7"]
        }
      }
    }
  ]
}
```

`aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, `aws:TagKeys`

タグベースの条件を指定します。詳細については、「タグを使用した Elastic Beanstalk リソースへのアクセスのコントロール」を参照してください。

タグを使用した Elastic Beanstalk リソースへのアクセスのコントロール

このトピックでは、タグベースのアクセスコントロールが IAM ポリシーの作成と管理にどのように役立つかについて説明します。

IAM ユーザーポリシーステートメントの条件を使用して、Elastic Beanstalk のリソースへのアクセス許可を設定できます。ポリシーステートメント条件の詳細については、「Elastic Beanstalk アクションのリソースと条件」を参照してください。条件内でタグを使用することは、リソースとリクエストへのアクセスをコントロールするひとつの方法です。Elastic Beanstalk リソースのタグ付けについては、「Elastic Beanstalk アプリケーションリソースのタグ付け」を参照してください。

IAM ポリシーの設計時に特定のリソースへのアクセス権を付与することで、詳細なアクセス許可を設定できます。管理するリソースの数が増えるに従って、このタスクはより困難になります。リソースにタグ付けしてポリシーステートメント条件でタグを使用することにより、このタスクをより容易にすることができます。特定のタグを使用して任意のリソースへのアクセス権を一括して付与します。次に、作成時や以降の段階で、このタグを関連リソースに繰り返し適用します。

タグは、リソースにアタッチしたり、タグ付けをサポートするサービスへのリクエストに渡したりすることができます。Elastic Beanstalk では、リソースにタグを付けることができ、一部のアクションにタグを含めることができます。IAM ポリシーを作成するときに、タグ条件キーを使用して次の条件をコントロールできます。

- 既にあるタグに基づいて、どのユーザーが環境に対してアクションを実行できるか。
- アクションのリクエストで渡すことができるタグ。
- リクエストで特定のタグキーを使用できるかどうか。

タグ条件キーの完全な構文と意味については、[IAM ユーザーガイド]の [タグを使用したアクセスコントロール]」を参照してください。

ポリシーのタグ条件の例

以下の例は、Elastic Beanstalk ユーザー用のポリシーでタグ条件を指定する方法を示しています。

Example 1: リクエストのタグに基づいてアクションを制限する

Elastic Beanstalk の管理ユーザーポリシー、AdministratorAccess-AWSElasticBeanstalk では、Elastic Beanstalk で管理されている任意のリソースに対して、任意の Elastic Beanstalk アクションを実行するための、無制限のアクセス許可をユーザーに付与します。

以下のポリシーでは、この権限を制限し、未認可のユーザーに対して Elastic Beanstalk の本番稼働環境を作成するアクセス許可を拒否します。これを行うには、リクエストに指定されているタグ `stage` の値が `gamma` または `prod` のいずれかである場合、`CreateEnvironment` アクションを拒否します。また、このポリシーでは、タグ変更アクションにこれらの同じタグ値を含めることや `stage` タグを完全に削除することを許可しないことで、これらの未承認のユーザーが本番稼働用環境のステージを改ざんするのを防ぎます。お客様の管理者は、未認証の IAM ユーザーには、マネージドユーザーポリシーに加えて、この IAM ポリシーをアタッチする必要があります。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk:AddTags"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/stage": ["gamma", "prod"]
        }
      }
    },
    {
      "Effect": "Deny",
```

```
      "Action": [
        "elasticbeanstalk:RemoveTags"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:TagKeys": ["stage"]
        }
      }
    }
  ]
}
```

Example 2: リソースタグに基づいてアクションを制限する

Elastic Beanstalk の管理ユーザーポリシー、AdministratorAccess-AWSElasticBeanstalk では、Elastic Beanstalk で管理されている任意のリソースに対して、任意の Elastic Beanstalk アクションを実行するための、無制限のアクセス許可をユーザーに付与します。

以下のポリシーでは、この権限を制限し、未認可のユーザーに対して Elastic Beanstalk 本番稼働環境でアクションを実行するアクセス許可を拒否します。これを行うには、環境に含まれているタグ `stage` の値が `gamma` または `prod` のいずれかである場合、特定のアクションを拒否します。お客様の管理者は、権限のない IAM ユーザーには、マネージドユーザーポリシーに加えて、この IAM ポリシーをアタッチする必要があります。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:AddTags",
        "elasticbeanstalk:RemoveTags",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticbeanstalk:TerminateEnvironment",
        "elasticbeanstalk:UpdateEnvironment",
        "elasticbeanstalk:ListTagsForResource"
      ],
      "Resource": "*",
```

```
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/stage": ["gamma", "prod"]
        }
      }
    }
  ]
}
```

Example 3: リクエストのタグに基づいてアクションを許可する

以下のポリシーでは、Elastic Beanstalk の開発アプリケーションを作成するアクセス許可をユーザーに付与します。

これを行うには、リクエストに指定されているタグ `stage` の値が `development` である場合、`CreateApplication` アクションと `AddTags` アクションを許可します。`aws:TagKeys` 条件により、ユーザーは他のタグキーを追加できなくなります。特に、`stage` タグキーの大文字と小文字が確実に区別されます。このポリシーは、Elastic Beanstalk の管理ユーザーポリシー、AdministratorAccess-AWSElasticBeanstalk が、IAM ユーザーにアタッチされていない場合に役立ちます。管理ポリシーでは、Elastic Beanstalk で管理される任意のリソースに対して任意の Elastic Beanstalk アクションを実行する無制限のアクセス許可をユーザーに付与します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk:AddTags"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/stage": "development"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["stage"]
        }
```

```
      }
    }
  ]
}
```

Example 4: リソースタグに基づいてアクションを制限する

以下のポリシーでは、Elastic Beanstalk の開発アプリケーションに対するアクションを実行し、これらのアプリケーションに関する情報を取得するアクセス許可をユーザーに付与します。

これを行うには、アプリケーションに含まれているタグ `stage` の値が `development` である場合に、特定のアクションを許可します。`aws:TagKeys` 条件により、ユーザーは他のタグキーを追加できなくなります。特に、`stage` タグキーの大文字と小文字が確実に区別されます。このポリシーは、Elastic Beanstalk の管理ユーザーポリシー、AdministratorAccess-AWSElasticBeanstalk が、IAM ユーザーにアタッチされていない場合に役立ちます。管理ポリシーでは、Elastic Beanstalk で管理される任意のリソースに対して任意の Elastic Beanstalk アクションを実行する無制限のアクセス許可をユーザーに付与します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:UpdateApplication",
        "elasticbeanstalk:DeleteApplication",
        "elasticbeanstalk:DescribeApplications"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/stage": "development"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["stage"]
        }
      }
    }
  ]
```

```
}
```

管理ポリシーに基づくポリシーの例

このセクションでは、 へのユーザーアクセスを制御する方法を示し AWS Elastic Beanstalk 、一般的なシナリオに必要なアクセスを提供するポリシーの例を示します。これらのポリシーは、Elastic Beanstalk 管理ポリシーから派生しています。ユーザーまたはグループに管理ポリシーをアタッチする方法については、「Elastic Beanstalk ユーザーポリシーの管理」を参照してください。

このシナリオでは、Example Corp. というソフトウェア会社の 3 つのチームが自社ウェブサイト関連の作業に携わっています。インフラストラクチャを管理する管理者チーム、ウェブサイト向けソフトウェアを開発する開発者チーム、そしてウェブサイトをテストする QA チームです。Elastic Beanstalk リソースに対するアクセス許可を管理しやすいように、Example Corp. では、Admins、Developers、Testers の 3 つグループを作成し、各チームのメンバーを追加します。Example Corp. は、Admins グループがすべての Elastic Beanstalk アセットの作成、トラブルシューティング、削除を行えるように、すべてのアプリケーション、環境、およびそれらの基盤となるリソースへのフルアクセスを許可したいと考えています。開発者には、すべての Elastic Beanstalk アセットの表示、およびアプリケーションバージョンの作成とデプロイを許可する必要があります。開発者がアプリケーションまたは環境を新規作成したり、実行中の環境を終了したりできないようにします。テスターは、アプリケーションをモニタリングおよびテストするために、すべての Elastic Beanstalk リソースを表示する必要があります。テスターは、Elastic Beanstalk リソースに変更を加えることができないようにする必要があります。

次のポリシーの例では、各グループに必要なアクセス権限を付与しています。

例 1: 管理者グループ - すべての Elastic Beanstalk および関連するサービス API

以下のポリシーにより、Elastic Beanstalk の使用に必要なすべてのアクションに対するアクセス許可がユーザーに付与されます。このポリシーでは、以下のサービスで Elastic Beanstalk がユーザーに代わってリソースのプロビジョニングと管理を行うことを許可しています。Elastic Beanstalk は、環境を作成するときに、これらの追加サービスに依存して基盤となるリソースをプロビジョニングします。

- Amazon Elastic Compute Cloud
- エラスティックロードバランシング
- Auto Scaling
- Amazon CloudWatch

- Amazon Simple Storage Service
- Amazon Simple Notification Service
- Amazon Relational Database Service
- AWS CloudFormation

このポリシーは例です。これにより、Elastic Beanstalk がアプリケーションと環境を管理するのに使用する AWS のサービスに、広範なアクセス許可のセットが付与されます。たとえば、 `ec2:*`は AWS Identity and Access Management (IAM) ユーザーが AWS アカウント内の任意の Amazon EC2 リソースに対して任意のアクションを実行できるようにします。これらのアクセス許可は、Elastic Beanstalk で使用するリソースに限定されません。ベストプラクティスとして、職務遂行に必要な許可のみを個人に付与することをお勧めします。

JSON

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "rds:*",
        "cloudformation:*"
      ],
      "Resource" : "*"
    }
  ]
}
```

例 2: 開発者グループ - 高い特権のオペレーションを除くすべてのオペレーション。

以下のポリシーでは、アプリケーションと環境を作成するアクセス許可を拒否し、それ以外の Elastic Beanstalk アクションをすべて許可します。

このポリシーは例です。これにより、Elastic Beanstalk がアプリケーションと環境を管理するのに使用する AWS 製品に、広範なアクセス許可のセットが付与されます。例えば、`ec2:*` を使用すると、IAM ユーザーが AWS アカウントで、すべての Amazon EC2 リソースに対してどんなアクションでも実行できます。これらのアクセス許可は、Elastic Beanstalk で使用するリソースに限定されません。ベストプラクティスとして、職務遂行に必要な許可のみを個人に付与することをお勧めします。

JSON

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Action" : [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk:DeleteApplication",
        "elasticbeanstalk:RebuildEnvironment",
        "elasticbeanstalk:SwapEnvironmentCNAMEs",
        "elasticbeanstalk:TerminateEnvironment"],
      "Effect" : "Deny",
      "Resource" : "*"
    },
    {
      "Action" : [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "rds:*",
        "cloudformation:*"],
      "Effect" : "Allow",
      "Resource" : "*"
    }
```

```
  ]
}
```

例 3: テスター - 表示のみ

次のポリシーは、すべてのアプリケーション、アプリケーションバージョン、イベント、および環境に対して読み取り専用アクセスを許可します。任意のアクションを実行することはできません。

JSON

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "elasticbeanstalk:Check*",
        "elasticbeanstalk:Describe*",
        "elasticbeanstalk:List*",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "ec2:Describe*",
        "elasticloadbalancing:Describe*",
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:List*",
        "cloudwatch:Get*",
        "s3:Get*",
        "s3:List*",
        "sns:Get*",
        "sns:List*",
        "rds:Describe*",
        "cloudformation:Describe*",
        "cloudformation:Get*",
        "cloudformation:List*",
        "cloudformation:Validate*",
        "cloudformation:Estimate*"
      ],
      "Resource" : "*"
    }
  ]
```

```
}
```

リソースに対するアクセス許可に基づいたポリシーの例

このセクションでは、特定の Elastic Beanstalk リソースにアクセスする Elastic Beanstalk アクションに対するユーザーのアクセス許可を制御するユースケースについて説明します。そのユースケースをサポートするサンプルポリシーについて説明します。Elastic Beanstalk リソースのポリシーの詳細については、「カスタムユーザーポリシーの作成」を参照してください。ユーザーおよびグループにポリシーをアタッチする方法については、 AWS Identity and Access Managementの使用の「IAM ポリシーの管理」を参照してください。

このユースケースの Example Corp. は小規模コンサルタント会社で、2 つのカスタマー用にアプリケーションを開発しています。開発マネージャーである John は、2 つの Elastic Beanstalk アプリケーション、app1 と app2 の開発を監督しています。John は 2 つのアプリケーションを開発し、テストをいくつか実行します。また、ジョンのみがこの 2 つのアプリケーションに対して本番環境を更新できます。ジョンが app1 および app2 に対して必要とする許可を次に示します。

- アプリケーション、アプリケーションバージョン、環境、および設定テンプレートを表示する
- アプリケーションバージョンを作成し、ステージング環境をデプロイする
- 本番環境を更新する
- 環境を作成および終了する

テスターである Jill には、アプリケーション、アプリケーションバージョン、環境、および設定テンプレートの 4 つのリソースを表示するためのアクセス許可が必要です。これにより、2 つのアプリケーションを監視およびテストします。ただし、Elastic Beanstalk リソースに変更を加えることはできません。

app1 の開発者である Jack には、app1 のすべてのリソースを表示するためのアクセス許可が必要です。また、app1 のアプリケーションバージョンを作成し、ステージング環境にデプロイする必要もあります。

Judy は Example Corp の AWS アカウントの管理者です。 John、Jill、Jack の IAM ユーザーを作成し、それらのユーザーに次のポリシーをアタッチして、app1 および app2 アプリケーションに適切なアクセス許可を付与します。

例 1: John - app1、app2 の開発マネージャー

確認および管理しやすいように John のポリシーを 3 つに分割しました。2 つのアプリケーションで John が開発、テスト、デプロイの各アクションを実行するための許可は、このすべてのポリシーによって付与されます。

最初のポリシーは、Auto Scaling、Amazon S3、Amazon EC2、CloudWatch、Amazon SNS、Elastic Load Balancing、Amazon RDS、および AWS CloudFormationのアクションを指定します。Elastic Beanstalk は、環境を作成するときに、これらの追加サービスに依存して基盤となるリソースをプロビジョニングします。

このポリシーは例です。これにより、Elastic Beanstalk がアプリケーションと環境を管理するために使用する AWS 製品に、幅広いアクセス許可が付与されます。たとえば、 `ec2:*`は IAM ユーザーが AWS アカウント内の任意の Amazon EC2 リソースに対して任意のアクションを実行できるようにします。これらのアクセス許可は、Elastic Beanstalk で使用するリソースに限定されません。ベストプラクティスとして、職務遂行に必要な許可のみを個人に付与することをお勧めします。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
   {
   "Effect": "Allow",
   "Action": [
   "ec2:*",
   "ecs:*",
   "ecr:*",
   "elasticloadbalancing:*",
   "autoscaling:*",
   "cloudwatch:*",
   "s3:*",
   "sns:*",
   "cloudformation:*",
   "dynamodb:*",
   "rds:*",
   "sqs:*",
   "logs:*",
   "iam:GetPolicyVersion",
   "iam:GetRole",
   "iam:ListRolePolicies",
```

```
    "iam:ListAttachedRolePolicies",
    "iam:ListInstanceProfiles",
    "iam:ListRoles",
    "iam:ListServerCertificates",
    "acm:DescribeCertificate",
    "acm:ListCertificates",
    "codebuild:CreateProject",
    "codebuild:DeleteProject",
    "codebuild:BatchGetBuilds",
    "codebuild:StartBuild"
    ],
    "Resource": "*"
    },
    {
    "Effect": "Allow",
    "Action": [
    "iam:PassRole"
    ],
    "Resource": "arn:aws:iam::111122223333:role/MyRole"
    }
    ]
    }
```

2 番目のポリシーでは、app1 および app2 リソースに対する John による実行を許可する Elastic Beanstalk アクションを指定します。`AllCallsInApplications` ステートメントでは、app1 および app2 内のすべてのリソース (`"elasticbeanstalk:*"` など) に対して実行されるすべての Elastic Beanstalk アクション (`elasticbeanstalk:CreateEnvironment` など) を許可します。`AllCallsOnApplications` ステートメントでは、app1 および app2 アプリケーションのリソース (`"elasticbeanstalk:*"`、`elasticbeanstalk:DescribeApplications` など) に対するすべての Elastic Beanstalk アクション (`elasticbeanstalk:UpdateApplication` など) を許可します。`AllCallsOnSolutionStacks` ステートメントでは、ソリューションスタックリソース (`"elasticbeanstalk:*"` など) に対するすべての Elastic Beanstalk アクション (`elasticbeanstalk:ListAvailableSolutionStacks` など) を許可します。

JSON

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

```
         "Sid":"AllCallsInApplications",
         "Action":[
            "elasticbeanstalk:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app1",
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app2"
               ]
            }
         }
      },
      {
         "Sid":"AllCallsOnApplications",
         "Action":[
            "elasticbeanstalk:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1",
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app2"
         ]
      },
      {
         "Sid":"AllCallsOnSolutionStacks",
         "Action":[
            "elasticbeanstalk:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2::solutionstack/*"
         ]
      }
   ]
}
```

3 番目のポリシーでは、2 番目のポリシーがそれらの Elastic Beanstalk アクションを完了するためにアクセス許可を必要とする Elastic Beanstalk アクションを指定します。`AllNonResourceCalls` ステートメントでは `elasticbeanstalk:CheckDNSAvailability` アクションを許可します。このアクションは、`elasticbeanstalk:CreateEnvironment` などのアクションを呼び出すときに必要です。また、`elasticbeanstalk:CreateStorageLocation` アクションも許可します。このアクションは、`elasticbeanstalk:CreateApplication`、`elasticbeanstalk:CreateEnvironment` などのアクションに必要です。

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability",
            "elasticbeanstalk:CreateStorageLocation"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      }
   ]
}
```

例 2: Jill - app1、app2 のテスター

確認および管理しやすいように Jill のポリシーを 3 つに分割しました。2 つのアプリケーションで Jill がテストとモニタリングアクションを実行するための許可は、このすべてのポリシーによって付与されます。

最初のポリシーでは`Describe*`、Auto Scaling、Amazon S3`List*`、Amazon EC2、CloudWatch、Amazon SNS、Elastic Load Balancing、Amazon RDS、および AWS CloudFormation （ レガシー以外のコンテナタイプの場合) に対する 、、および `Get*`アクションを指定して、Elastic Beanstalk アクションが app1 および app2 アプリケーションの基盤となるリソースに関する関連情報を取得できるようにします。

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "ec2:Describe*",
            "elasticloadbalancing:Describe*",
            "autoscaling:Describe*",
            "cloudwatch:Describe*",
            "cloudwatch:List*",
            "cloudwatch:Get*",
            "s3:Get*",
            "s3:List*",
            "sns:Get*",
            "sns:List*",
            "rds:Describe*",
            "cloudformation:Describe*",
         "cloudformation:Get*",
         "cloudformation:List*",
         "cloudformation:Validate*",
         "cloudformation:Estimate*"
         ],
         "Resource":"*"
      }
   ]
}
```

2 番目のポリシーでは、app1 および app2 リソースに対する Jill による実行を許可する Elastic Beanstalk アクションを指定します。`AllReadCallsInApplications` ステートメントでは、Jill による `Describe*` アクションと環境情報アクションの呼び出しを許可します。`AllReadCallsOnApplications` ステートメントでは、Jill に app1 および app2 アプリケーションのリソースに対する `DescribeApplications` および `DescribeEvents` アクションの呼び出しを許可します。`AllReadCallsOnSolutionStacks` ステートメントでは、ソリューションスタックリソースに関連するアクション（`ListAvailableSolutionStacks`、`DescribeConfigurationOptions`、および `ValidateConfigurationSettings`）の表示を許可します。

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllReadCallsInApplications",
         "Action":[
            "elasticbeanstalk:Describe*",
            "elasticbeanstalk:RequestEnvironmentInfo",
            "elasticbeanstalk:RetrieveEnvironmentInfo"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app1",
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app2"
               ]
            }
         }
      },
      {
         "Sid":"AllReadCallsOnApplications",
         "Action":[
            "elasticbeanstalk:DescribeApplications",
            "elasticbeanstalk:DescribeEvents"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1",
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app2"
         ]
      },
      {
         "Sid":"AllReadCallsOnSolutionStacks",
         "Action":[
            "elasticbeanstalk:ListAvailableSolutionStacks",
```

```
            "elasticbeanstalk:DescribeConfigurationOptions",
            "elasticbeanstalk:ValidateConfigurationSettings"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2::solutionstack/*"
         ]
      }
   ]
}
```

3 番目のポリシーでは、2 番目のポリシーがそれらの Elastic Beanstalk アクションを完了するためにアクセス許可を必要とする Elastic Beanstalk アクションを指定します。`AllNonResourceCalls` ステートメントでは `elasticbeanstalk:CheckDNSAvailability` アクションを許可します。このアクションは、一部の表示アクションに必要です。

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      }
   ]
}
```

例 3: Jack - app1 の開発者

確認および管理しやすいように Jack のポリシーを 3 つに分割しました。app1 リソースで Jack が、テスト、モニタリング、デプロイの各アクションを実行するための許可は、このすべてのポリシーによって付与されます。

最初のポリシーでは、Auto Scaling、Amazon S3、Amazon EC2、CloudWatch、Amazon SNS、Elastic Load Balancing、Amazon RDS、および AWS CloudFormation（レガシー以外のコンテナタイプの場合) のアクションを指定して、Elastic Beanstalk アクションが app1 の基盤となるリソースを表示して操作できるようにします。サポートされているレガシーではないコンテナタイプのリストについては、「the section called “一部のプラットフォームバージョンがレガシーとマークされているのはなぜですか?”」を参照してください。

このポリシーは例です。これにより、Elastic Beanstalk がアプリケーションと環境を管理するために使用する AWS 製品に、幅広いアクセス許可が付与されます。たとえば、 `ec2:*` は IAM ユーザーが AWS アカウント内の任意の Amazon EC2 リソースに対して任意のアクションを実行できるようにします。これらのアクセス許可は、Elastic Beanstalk で使用するリソースに限定されません。ベストプラクティスとして、職務遂行に必要な許可のみを個人に付与することをお勧めします。

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "ec2:*",
            "elasticloadbalancing:*",
            "autoscaling:*",
            "cloudwatch:*",
            "s3:*",
            "sns:*",
            "rds:*",
            "cloudformation:*"
         ],
         "Resource":"*"
      }
   ]
}
```

2 番目のポリシーでは、app1 リソースに対する Jack による実行を許可する Elastic Beanstalk アクションを指定します。

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllReadCallsAndAllVersionCallsInApplications",
         "Action":[
            "elasticbeanstalk:Describe*",
            "elasticbeanstalk:RequestEnvironmentInfo",
            "elasticbeanstalk:RetrieveEnvironmentInfo",
            "elasticbeanstalk:CreateApplicationVersion",
            "elasticbeanstalk:DeleteApplicationVersion",
            "elasticbeanstalk:UpdateApplicationVersion"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app1"
               ]
            }
         }
      },
      {
         "Sid":"AllReadCallsOnApplications",
         "Action":[
            "elasticbeanstalk:DescribeApplications",
            "elasticbeanstalk:DescribeEvents"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1"
         ]
      },
      {
         "Sid":"UpdateEnvironmentInApplications",
         "Action":[
            "elasticbeanstalk:UpdateEnvironment"
```

```
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/app1/
app1-staging*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app1"
               ]
            },
            "StringLike":{
               "elasticbeanstalk:FromApplicationVersion":[
                  "arn:aws:elasticbeanstalk:us-
east-2:123456789012:applicationversion/app1/*"
               ]
            }
         }
      },
      {
         "Sid":"AllReadCallsOnSolutionStacks",
         "Action":[
            "elasticbeanstalk:ListAvailableSolutionStacks",
            "elasticbeanstalk:DescribeConfigurationOptions",
            "elasticbeanstalk:ValidateConfigurationSettings"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2::solutionstack/*"
         ]
      }
   ]
}
```

3 番目のポリシーでは、2 番目のポリシーがそれらの Elastic Beanstalk アクションを完了するためにアクセス許可を必要とする Elastic Beanstalk アクションを指定します。`AllNonResourceCalls` ステートメントでは `elasticbeanstalk:CheckDNSAvailability` アクションを許可します。このアクションは、`elasticbeanstalk:CreateEnvironment` などのアクションを呼び出すときに必

要です。また、`elasticbeanstalk:CreateStorageLocation` アクションも許可します。このアクションは、`elasticbeanstalk:CreateEnvironment` などのアクションに必要です。

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability",
            "elasticbeanstalk:CreateStorageLocation"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      }
   ]
}
```

環境間の Amazon S3 バケットアクセスの防止

このトピックでは、管理ポリシーが環境間の S3 バケットアクセスを許可する場合がある状況と、このタイプのアクセスを管理するカスタムポリシーを作成する方法について説明します。

Elastic Beanstalk には、 AWS アカウントの Elastic Beanstalk 環境に必要な AWS リソースを処理するためのマネージドポリシーが用意されています。 AWS アカウント内の 1 つのアプリケーションにデフォルトで提供されるアクセス許可は、同じ AWS アカウントの他のアプリケーションに属する S3 リソースにアクセスできます。

AWS アカウントが複数の Beanstalk アプリケーションを実行している場合、環境ごとに独自のサービスロールまたはインスタンスプロファイルにアタッチする独自のカスタムポリシーを作成することで、ポリシーのセキュリティを絞り込むことができます。そうすると、カスタムポリシーの S3 アクセス許可を特定の環境に制限できます。

> **Note**
>
> カスタムポリシーをメンテナンスする責任はお客様にあることに注意してください。カスタムポリシーのベースになっている Elastic Beanstalk 管理ポリシーが変更された場合は、ベースのポリシーに対する各変更についてカスタムポリシーを変更する必要があります。Elastic Beanstalk 管理ポリシーの変更履歴については、「Elastic Beanstalk の AWS マネージドポリシーの更新」を参照してください。

スコープダウンされたアクセス許可の例

次の例は、AWSElasticBeanstalkWebTier 管理ポリシーに基づいています。

デフォルトのポリシーには、S3 バケットへのアクセス許可に次の行が含まれます。このデフォルトポリシーは、S3 バケットアクションを特定の環境またはアプリケーションに制限しません。

```
{
   "Sid" : "BucketAccess",
   "Action" : [
      "s3:Get*",
      "s3:List*",
      "s3:PutObject"
     ],
   "Effect" : "Allow",
   "Resource" : [
      "arn:aws:s3:::elasticbeanstalk-*",
      "arn:aws:s3:::elasticbeanstalk-*/*"
     ]
}
```

特定のリソースを `Principal` として指定されたサービスロールに認定することで、アクセスをスコープダウンできます。次の例では、ID `my-example-env-ID` の環境内の S3 バケットに対するカスタムサービスロール `aws-elasticbeanstalk-ec2-role-my-example-env` のアクセス許可を示します。

Example 特定の環境の S3 バケットにのみアクセス許可を付与する

```
{
   "Sid": "BucketAccess",
   "Action": [
```

```
      "s3:Get*",
      "s3:List*",
      "s3:PutObject"
    ],
   "Effect": "Allow",
   "Principal": {
      "AWS": "arn:aws:iam::...:role/aws-elasticbeanstalk-ec2-role-my-example-env"
     },
   "Resource": [
      "arn:aws:s3:::elasticbeanstalk-my-region-account-id-12345",
      "arn:aws:s3:::elasticbeanstalk-my-region-account-id-12345/resources/environments/
my-example-env-ID/*"
    ]
}
```

> **Note**
>
> リソース ARN には (環境名ではなく) Elastic Beanstalk 環境 ID を含める必要があります。環境 ID は、Elastic Beanstalk コンソールの [環境の概要] ページから取得できます。 AWS CLI describe-environments コマンドを使用してこの情報を取得することもできます。

Elastic Beanstalk 環境の S3 バケットのアクセス許可を更新する方法の詳細については、次のリソースを参照してください。

- このガイドの「Amazon S3 で Elastic Beanstalk を使用する」
- 「サービス認証リファレンス」ガイドの「Amazon S3 で定義されるリソースタイプ」
- 「IAM ユーザーガイド」の「ARN 形式」

Amazon RDS で Elastic Beanstalk を使用する

このセクションでは、Elastic Beanstalk を Amazon Relational Database Service (Amazon RDS) とともに使用して、リレーショナルデータベースをセットアップ、運用、スケーリングする方法について説明します。設定に関するいくつかの概念を説明し、推奨事項を示します。次に、作成して Amazon RDS に接続するプロセスについて説明します。

開始するには 2 つのオプションがあります。

- Amazon RDS に新しい データベースを作成する。

- 以前 Elastic Beanstalk で作成して、後に Beanstalk 環境からデカップリングされたデータベースから開始する。詳細については、「the section called “データベース”」を参照してください。

アプローチを選択する

どちらの方法でも、Amazon RDS でデータベースインスタンスを実行し、アプリケーションの起動時にそれに接続するように設定できます。複数の環境をデータベースに接続できます。

> **Note**
>
> アプリケーションでデータベースインスタンスを使用したことがない場合は、最初に Elastic Beanstalk コンソールを使用してテスト環境にデータベースを追加することをお勧めします。これにより、スタンドアロンデータベースに必要な追加の設定作業をせずに、アプリケーションで環境プロパティを読み取り、接続文字列を作成し、データベースインスタンスに接続できることを確認できます。詳細については、「Elastic Beanstalk 環境にデータベースを追加する」を参照してください。

セキュリティグループを設定する

環境内で Amazon EC2 インスタンスに外部データベースへの接続を許可するために、環境に関連付けられた Auto Scaling グループに追加のセキュリティグループを設定できます。データベースインスタンスにアタッチされているのと同じセキュリティグループをアタッチできます。または、別のセキュリティグループを使用することもできます。別のセキュリティグループをアタッチする場合は、データベースにアタッチされているセキュリティグループを設定して、このセキュリティグループからのインバウンドアクセスを許可する必要があります。

> **Note**
>
> データベースにアタッチされているセキュリティグループにルールを追加することで、データベースに環境を接続できます。このルールは、Elastic Beanstalk が環境の Auto Scaling グループにアタッチする自動生成されたセキュリティグループからのインバウンドアクセスを許可する必要があります。ただし、このルールを作成することで 2 つのセキュリティグループ間に依存性が生じます。データベースのセキュリティグループは環境のセキュリティグループに依存するため、その後、環境を終了しようとすると、Elastic Beanstalk は環境のセキュリティグループを削除できなくなります。

データベース接続を設定する

データベースインスタンスを起動しセキュリティグループを設定すると、環境プロパティを使用して、エンドポイントやパスワードなどの接続情報をアプリケーションに渡すことができます。これは、ご使用の環境でデータベースインスタンスを実行するときにバックグラウンドで Elastic Beanstalk が使用するのと同じメカニズムです。

セキュリティのレイヤーを追加するには、接続情報を Amazon S3 に保存し、デプロイの間にデータを取得するように Elastic Beanstalk を設定します。設定ファイル `.ebextensions` () を使用して環境内のインスタンスを設定し、アプリケーションをデプロイするときに、Amazon S3 からファイルを安全に取得できます。

トピック

- デフォルトの VPC で外部 Amazon RDS インスタンスを起動して接続
- での Amazon RDS 認証情報の保存 AWS Secrets Manager
- 外部 Amazon RDS インスタンスのクリーンアップ

デフォルトの VPC で外部 Amazon RDS インスタンスを起動して接続

次の手順では、外部の Amazon RDS インスタンスをデフォルトの VPC に接続するプロセスについて説明します。このプロセスは、カスタム VPC を使用する場合も同じです。唯一の追加要件となるのが、環境と DB インスタンスが相互に通信可能な同じサブネット内に存在することです。カスタム VPC を Elastic Beanstalk で設定する方法の詳細については、「Amazon VPC で Elastic Beanstalk を使用する」を参照してください。

> **Note**
>
> - 新しい DB インスタンスを起動する代わりに、以前に Elastic Beanstalk で作成して、その後、Beanstalk 環境からデカップリングされたデータベースから開始することもできます。詳細については、「the section called “データベース”」を参照してください。このオプションを使用すると、新しいデータベースを起動する手順を行う必要はありません。ただし、このトピックで説明するその後の手順を完了する必要があります。
> - Elastic Beanstalk によって作成され、その後 Beanstalk 環境からデカップリングされたデータベースから開始する場合は、最初のグループの手順をスキップして、「RDS インスタンスのセキュリティグループのインバウンドルールを変更するには」以下のグループの手順から続行できます。

- デカップリングしたデータベースを実稼働環境で使用する場合は、データベースが使用するストレージタイプがワークロードに適していることを確認します。詳細については、Amazon RDS ユーザーガイドの「Amazon RDS DB インスタンスストレージ」および「Amazon RDS DB インスタンスを変更する」を参照してください。

RDS DB インスタンスをデフォルト VPC 内で起動するには

1. RDS コンソールを開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. [データベースの作成] を選択します。
4. [Standard Create (スタンダード作成)] を選択します。

> **Important**
>
> [Easy Create (簡易作成)] を選択しないでください。これを選択した場合、この RDS DB の起動に必要な設定ができません。

5. [Additional configuration (追加の設定)] の [Initial database name (初期データベース名)] に「**ebdb**」と入力します。
6. デフォルト設定を確認し、特定の要件に従ってこれらの設定を調整します。以下のオプションに注目します。
   - DB instance class (DB インスタンスクラス) – ワークロードに適したメモリ量と CPU 能力があるインスタンスサイズを選択します。
   - Multi-AZ deployment (マルチ AZ 配置) – 高可用性を得るには、これを [Create an Aurora Replica/Reader node in a different AZ (異なる AZ に Aurora レプリカ/リーダーノードを作成)] に設定します。
   - [Master username (マスター・ユーザーネーム)] と [Master password (マスターパスワード)] – データベースのユーザー名とパスワード。後で使用するため、これらの設定を書き留めておきます。
7. 残りのオプションのデフォルト設定を確認し、[データベースの作成] を選択します。

次に、DB インスタンスにアタッチするセキュリティグループを変更して、適切なポートへのインバウンドトラフィックを許可します。これは、後で Elastic Beanstalk 環境にアタッチするのと同じセ

キュリティグループです。その結果、追加するルールは、同じセキュリティグループ内の他のリソースにインバウンドアクセスのアクセス許可を付与します。

RDS インスタンスにアタッチされているセキュリティグループのインバウンドルールを変更するには

1. Amazon RDS コンソールを開きます。
2. [データベース] を選択します。
3. 詳細を表示する DB インスタンスの名前を選択します。
4. [Connectivity] (接続) セクションで、このページに表示される [Subnets] (サブネット)、[Security groups] (セキュリティグループ)、[Endpoint] (エンドポイント) をメモします。これは、後でこの情報を使用できるようにするためです。
5. [Security] (セキュリティ) には、DB インスタンスに関連付けられるセキュリティグループが表示されます。リンクを開いて、Amazon EC2 コンソールにセキュリティグループを表示します。
6. セキュリティグループの詳細で、インバウンド を選択します。
7. [編集] を選択します。
8. [ルールの追加] を選択します。
9. タイプ として、アプリケーションが使用する DB エンジンを選択します。
10. 出典 として、**sg-** と入力して、使用可能なセキュリティグループのリストを表示します。Elastic Beanstalk 環境で使用される Auto Scaling グループに関連付けられているセキュリティグループを選択します。これは、環境内の Amazon EC2 インスタンスがデータベースにアクセスできるようにするためです。
11. [保存] を選択します。

次に、実行中の環境に DB インスタンスのセキュリティグループを追加します。この手順では、アタッチされる追加のセキュリティグループを使用して、Elastic Beanstalk が環境内のすべてのインスタンスの再プロビジョニングを行います。

環境にセキュリティグループを追加するには

- 次のいずれかを行います：
  - Elastic Beanstalk コンソールを使用してセキュリティグループを追加するには

a. Elastic Beanstalk 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. ナビゲーションペインで、[環境] を選択し、リストから環境の名前を選択します。
c. ナビゲーションペインで、[設定] を選択します。
d. [インスタンス] 設定カテゴリで、[編集] を選択します。
e. EC2 セキュリティグループで、Elastic Beanstalk が作成するインスタンスセキュリティグループに加えて、インスタンスにアタッチするセキュリティグループを選択します。
f. ページの最下部で [適用] を選択し変更を保存します。
g. 警告を読み取り、確認 を選択します。

- 設定ファイルを使用してセキュリティグループを追加するには、`securitygroup-addexisting.config` サンプルファイルを使用します。

次に、環境プロパティを使用して環境に接続情報を渡します。Elastic Beanstalk コンソールを使用して DB インスタンスを環境に追加すると、Elastic Beanstalk は [RDS_HOSTNAME] など環境プロパティを使用して、アプリケーションに接続情報を渡します。同じプロパティを使用できます。これにより、統合 DB インスタンスと外部 DB インスタンスの両方で同じアプリケーションコードを使用できます。または、独自のプロパティ名を選択することもできます。

Amazon RDS DB インスタンスの環境プロパティを設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. [環境プロパティ] セクションで、アプリケーションが読み取りする変数を定義して、接続文字列を構成します。統合された RDS DB インスタンスがある環境との互換性を考慮して、以下の名前と値を使用します。パスワードを除くすべての値は、RDS コンソールで見つかります。

プロパティ名	説明	プロパティ値
`RDS_HOSTNAME`	DB インスタンスのホスト名。	Amazon RDS コンソールの [Connectivity & security (Connectivityとセキュリ

プロパティ名	説明	プロパティ値
		ティ)] タブ: [Endpoint (エンドポイント)]。
RDS_PORT	DB インスタンスが接続を許可するポート。デフォルト値は DB エンジンによって異なります。	Amazon RDS コンソールの [Connectivity & security (接続とセキュリティ)] タブ: [Port (ポート)]。
RDS_DB_NAME	データベース名 **ebdb**。	Amazon RDS コンソールの [Configuration (設定)] タブ: [DB Name (DB 名)]。
RDS_USERNAME	お客様のデータベース用に設定したユーザー名。	Amazon RDS コンソールの [Configuration (設定)] タブ: [Master username (マスターユーザー名)]。
RDS_PASSWORD	お客様のデータベース用に設定したパスワード。	Amazon RDS コンソールではリファレンスできません。

6. ページの最下部で [適用] を選択し変更を保存します。

アプリケーションが環境プロパティを読み取り、接続文字列を作成するようにまだプログラムしていない場合は、次の言語固有のトピックで手順を参照してください。

- Java SE – データベースへの接続 (Java SE プラットフォーム)
- Java と Tomcat – データベースへの接続 (Tomcat プラットフォーム)
- Node.js – データベースへの接続
- .NET – データベースへの接続
- PHP – PDO または MySQLi を使用してデータベースに接続
- Python – データベースへの接続
- Ruby – データベースへの接続

最後に、環境変数を読み込むアプリケーションによっては、環境のインスタンス上でアプリケーションサーバーを再起動する必要があります。

環境のアプリケーションサーバーを再起動するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. [Actions] (アクション) を選択してから、[Restart app server(s)] (アプリサーバーの再起動) を選択します。

での Amazon RDS 認証情報の保存 AWS Secrets Manager

このトピックでは、 AWS Secrets Manager が Elastic Beanstalk で認証情報を取得するためのセキュリティ体制を改善する方法について説明します。また、Elastic Beanstalk アプリケーションの認証情報を設定するのに役立つ特定のリソースへの参照も提供します。

AWS Secrets Manager は、暗号化された認証情報を保存および取得する機能を提供することで、セキュリティ体制の改善に役立ちます。認証情報を Secrets Manager に保存することで、アプリケーションまたはそれに関連するコンポーネントを調べることができるすべてのユーザーによる侵害の可能性を回避できます。コードは Secrets Manager サービスをランタイムに呼び出して、認証情報を動的に取得できます。Secrets Manager には、Python、Go、Java などのランタイム言語用のクライアント側シークレットキャッシュコンポーネントなどの機能も用意されています。

詳細については、AWS Secrets Manager ユーザーガイドの次のトピックを参照してください。

- Amazon RDS が AWS Secrets Manager を使用する方法
- AWS Secrets Manager データベースシークレットを作成する
- からシークレットを取得する AWS Secrets Manager

外部 Amazon RDS インスタンスのクリーンアップ

外部 Amazon RDS インスタンスを Elastic Beanstalk 環境に接続すると、データベースインスタンスは環境のライフサイクルに依存しないため、環境を終了しても削除されません。データベースインスタンスに保存した個人情報が不必要に保持されないようにするには、不要になったレコードはすべて削除します。または、データベースインスタンスを削除します。

Amazon S3 で Elastic Beanstalk を使用する

このトピックでは、Elastic Beanstalk が Amazon Simple Storage Service (Amazon S3) を使用する方法と、S3 バケットに保存するオブジェクトのタイプについて説明します。また、Elastic Beanstalk 環境を終了した後に手動で削除する必要があるオブジェクトについても記載し、その手順を示します。

Elastic Beanstalk Amazon S3 カスタマーアカウントバケット

Elastic Beanstalk は、環境を作成した各リージョンに対して、`elasticbeanstalk-`*`region`*`-`*`account-id` という名前の Amazon S3 バケットを作成して暗号化します。 AWS アカウントはこのバケットを所有しています。Elastic Beanstalk は、アプリケーションの適切なオペレーションのために、一時的な設定ファイルやその他のオブジェクトをこのバケットに保存します。Elastic Beanstalk では、サービスマネージドバケットに対して有効な ACLs が必要であるため、このバケットのアクセスコントロールリスト (ACL) を有効にします。

Amazon S3 はデフォルトでバケットアクセスコントロールリスト (ACLsを無効にすることに注意してください。さらに、Amazon S3 ユーザーガイドの ACL 概要トピックでは、特定のユースケースを除き、ACLs無効にしておくことをお勧めします。Elastic Beanstalk のサービスマネージドバケットは、有効な ACLs。セキュリティを維持するために、Elastic Beanstalk デプロイでは、このバケットがアプリケーションを実行しているアカウントによって所有されていることを強制します。

Elastic Beanstalk は、Amazon S3 バケットによって提供されるデフォルトの暗号化を保持します。バケット暗号化の詳細については、Amazon S3 のデフォルトの暗号化」を参照してください。

Elastic Beanstalk Amazon S3 カスタマーアカウントバケットの内容

次の表に、Elastic Beanstalk がカスタマーアカウントバケットに保存するいくつかのオブジェクトを示します。この表には、手動で削除する必要があるオブジェクトも示されています。不要なストレージコストを避け、個人情報が保持されないようにするために、不要になったオブジェクトを手動で削除してください。

オブジェクト	いつ保存されますか。	いつ削除されますか。
アプリケーションバージョン	環境を作成したり、既存の環境にアプリケーションコードをデプロイすると、Elastic Beanstalk はアプリケーションバー	アプリケーションの削除中は、「バージョンライフサイクル」に従います。

オブジェクト	いつ保存されますか。	いつ削除されますか。
	ジョンを Amazon S3 に保存し、それを環境に関連付けます。	
ソースバンドル	Elastic Beanstalk コンソールまたは EB CLI を使用して新しいアプリケーションバージョンをアップロードすると、Elastic Beanstalk はそのコピーを Amazon S3 に保管し、環境のソースバンドルとして設定します。	手動。アプリケーションバージョンを削除すると、[Amazon S3 からのバージョンの削除] を選択して、関連するソースバンドルを削除することもできます。詳細については、「アプリケーションバージョンの管理」を参照してください。
カスタムプラットフォーム	カスタムプラットフォームを作成すると、Elastic Beanstalk は関連するデータを一時的に Amazon S3 に保存します。	カスタムプラットフォームの作成が正常に完了したとします。
ログファイル	Elastic Beanstalk にインスタンスログファイル (テールログまたはバンドルログ) を取得して Amazon S3 に保存するようにリクエストできます。ログのローテーションを有効にし、ローテーション後にログを自動的に Amazon S3 に発行するように環境を設定することもできます。	ログ末尾およびバンドルログ: 作成の 15 分後に削除されます。 ローテーションされたログ: 手動。
保存された設定	手動。	手動。

Elastic Beanstalk Amazon S3 バケット内のオブジェクトの削除

環境を終了するかアプリケーションを削除すると、Elastic Beanstalk は Amazon S3 から大部分の関連オブジェクトを削除します。実行中のアプリケーションのストレージコストを最小限に抑えるには、アプリケーションが必要としないオブジェクトを定期的に削除します。さらに、「Elastic Beanstalk Amazon S3 カスタマーアカウントバケットの内容」にリストされているように、手動で削除する必要のあるオブジェクトに注意してください。個人情報が不必要に保持されないようにするには、不要になった時点でそれらのオブジェクトを削除します。

- アプリケーションで使用する予定のないアプリケーションのバージョンを削除してください。アプリケーションバージョンを削除すると、[Amazon S3 からのバージョンの削除] を選択して関連ソースバンドルを削除することもできます。ソースバンドルはアプリケーションのソースコードと設定ファイルのコピーで、アプリケーションをデプロイしたり、アプリケーションのバージョンをアップロードしたりしたときに、Elastic Beanstalk が Amazon S3 にアップロードしたものです。アプリケーションバージョンを削除する方法については、「アプリケーションバージョンの管理」を参照してください。
- ローテーションされた不要なログを削除します。または、それらをダウンロードするか、Amazon S3 Glacier に移動して、詳細に分析します。
- どの環境でも使用する予定のない保存済みの設定は、削除します。

Elastic Beanstalk Amazon S3 バケットの削除

Elastic Beanstalk がバケットを作成すると、新しいバケットに適用されるバケットポリシーも作成されます。このポリシーには次の 2 つの目的があります。

- 環境がバケットに書き込めるようにすること。
- バケットが誤って削除されるのを防ぐこと。

Elastic Beanstalk が環境用に作成したバケットに適用するポリシーにより、最初に意図的にバケットポリシーを削除しない限り、これらのバケットを削除することはできません。Amazon S3 コンソールでバケットプロパティの [アクセス許可] セクションからバケットポリシーを削除できます。

> **Warning**
>
> バケット全体を削除するのではなく、特定の不要なオブジェクトを Elastic Beanstalk Amazon S3 バケットから削除することをお勧めします。
> アカウントで Elastic Beanstalk が作成したバケットを削除しても、既存のアプリケーションと実行中の環境が対応するリージョンに残っていると、アプリケーションが正しく機能しなくなる可能性があります。例:
>
> - 環境がスケールアウトされると、Elastic Beanstalk は Amazon S3 バケットで環境のアプリケーションバージョンを見つけて、新しい Amazon EC2 インスタンスを開始するためにそれを使用できるはずです。
> - カスタムプラットフォームを作成すると、Elastic Beanstalk は作成プロセス中に一時ストレージおよび Amazon S3 ストレージを使用します。

S3 バケットを削除する意味の詳細については、「Amazon S3 ユーザーガイド」の「バケットの削除」に記載されている考慮事項を参照してください。

Elastic Beanstalk ストレージバケットを削除するには (コンソール)

S3 バケットを削除する一般的な手順については、「Amazon S3 ユーザーガイド」の「バケットの削除」にも記載されています。以下の手順では、Elastic Beanstalk によって作成されたバケットを削除するため、最初にバケットポリシーを削除する追加ステップが含まれます。

1. Amazon S3 コンソールを開きます。
2. バケット名を選択して、Elastic Beanstalk ストレージバケットのページを開きます。
3. [アクセス許可] タブを選択します。
4. [バケットポリシー] を選択します。
5. [削除] を選択します。
6. Amazon S3 コンソールのメインページに戻り、Elastic Beanstalk ストレージバケットを選択します。
7. [Delete Bucket] (バケットの削除) を選択します。
8. テキストフィールドにバケット名を入力することでバケットを削除することを確認し、[バケットの削除] を選択します。

AWS Secrets Manager および Parameter Store での Elastic Beanstalk AWS Systems Manager の使用

このトピックでは、Elastic Beanstalk 環境で AWS Secrets Manager と AWS Systems Manager Parameter Store を使用して、認証情報や API キーなどの機密情報を安全に保存および取得する方法について説明します。アプリケーションは、これらのサービスの APIs またはコマンドラインツールを使用して、保存されたシークレットとパラメータをこれらのストアから直接取得できます。

Elastic Beanstalk では、環境変数で Secrets Manager および Systems Manager Parameter Store データを参照することもできます。これは、アプリケーションが API コールを管理することなく、これらのサービスに保存されているシークレットとパラメータにネイティブにアクセスする安全なオプションです。

トピック

- Elastic Beanstalk 環境変数へのシークレットとパラメータの取得
- Elastic Beanstalk がシークレットとパラメータにアクセスするために必要な IAM アクセス許可
- AWS Secrets Manager および AWS Systems Manager パラメータストアの使用
- シークレットと Elastic Beanstalk 環境変数の統合のトラブルシューティング

Elastic Beanstalk 環境変数へのシークレットとパラメータの取得

Elastic Beanstalk は、インスタンスのブートストラップ中に AWS Secrets Manager および AWS Systems Manager Parameter Store から値を取得し、アプリケーションが使用する環境変数に割り当てることができます。

以下のポイントは、環境変数をシークレットとして使用するための設定、同期、アクセスをまとめたものです。

- 保存するシークレットとパラメータの Amazon リソースネーム (ARNs) を指定して、シークレットを保存するように環境変数を設定します。
- Secrets Manager または Systems Manager パラメータストアでシークレット値を更新またはローテーションする場合は、環境変数を手動で更新する必要があります。
- シークレット環境変数は、ebextension コンテナコマンドとプラットフォームフックで使用できます。

サポートされるプラットフォームのバージョン

2025 年 3 月 26 日以降にリリースされたプラットフォームバージョンは、環境変数として設定された AWS Secrets Manager シークレットと Parameter Store AWS Systems Manager パラメータをサポートしています。

> **Note**
>
> Docker および ECS ベースの Docker プラットフォームを除き、Amazon Linux 2 プラットフォームバージョンは複数行変数値をサポートしていません。複数行変数のサポートの詳細については、「」を参照してください複数行の値。

トピック

- 料金

- シークレットを Elastic Beanstalk 環境変数として設定する
- Elastic Beanstalk 環境変数とのシークレット同期のベストプラクティス
- Amazon Linux 2 環境変数の複数行の値

料金

Secrets Manager と Systems Manager パラメータストアの使用には、標準料金が適用されます。料金の詳細については、以下のウェブサイトを参照してください。

- AWS Secrets Manager 料金
- AWS Systems Manager 料金 (コンテンツリストから Parameter Store を選択)

Elastic Beanstalk では、アプリケーションが環境変数を介して環境シークレットを参照するための料金は発生しません。ただし、Elastic Beanstalk がユーザーに代わってこれらのサービスに対して行うリクエストには、標準料金が適用されます。

シークレットを Elastic Beanstalk 環境変数として設定する

Elastic Beanstalk コンソール、 の設定ファイル`.ebextensions`、 CLI、および AWS SDK AWS を使用して、シークレットとパラメータを環境変数として設定できます。

トピック

- 前提条件
- コンソールを使用する
- .ebextensions のファイルを使用した設定
- を使用した設定 AWS CLI
- AWS SDK を使用した設定

前提条件

シークレットを参照するように環境変数を設定する前に、まず次のステップを完了する必要があります。

環境変数設定前の一般的な手順

1. Secrets Manager シークレットまたは Parameter Store パラメータを作成して、機密データを保存します。詳細については、次のトピックの 1 つまたは両方を参照してください。

- でのシークレットの作成 the section called “Secrets Manager の使用”
- でのパラメータの作成 the section called “Systems Manager パラメータストアの使用”

2. シークレットとパラメータを取得するために、環境の EC2 インスタンスに必要な IAM アクセス許可を設定します。詳細については、「必要な IAM 許可」を参照してください。

コンソールを使用する

Elastic Beanstalk コンソールを使用して、シークレットを環境変数として設定できます。

Elastic Beanstalk コンソールでシークレットを環境変数として設定するには

1. Elastic Beanstalk 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、[環境] を選択し、リストから環境の名前を選択します。
3. ナビゲーションペインで、[設定] を選択します。
4. [更新、モニタリング、ログ] の設定カテゴリで、[編集] を選択します。
5. Runtime 環境変数まで下にスクロールします。
6. 環境変数の追加 を選択します。
7. Source には、Secrets Manager または SSM パラメータストアを選択します。

> **Note**
> ドロップダウンのプレーンテキストオプションの詳細については、「」を参照してください環境プロパティ (環境変数) の設定。

8. 環境変数名 シークレットまたはパラメータ値を保持する環境変数の名前を入力します。
9. 環境変数値には、Systems Manager パラメータストアパラメータの ARN または Secrets Manager シークレットを入力します。インスタンスのブートストラップ中に、Elastic Beanstalk はステップ 8 で入力した変数の値を、この ARN リソースに保存されている値に開始します。

   コンソールは、入力した値がステップ 7 で選択したストアの有効な ARN 形式であるかどうかを検証します。ただし、ARN で指定されたリソースの存在や、それにアクセスするために必要な IAM アクセス許可があるかどうかは検証されません。
10. さらに変数を追加する必要がある場合は、ステップ 6 からステップ 9 を繰り返します。
11. ページの最下部で [適用] を選択し変更を保存します。

.ebextensions のファイルを使用した設定

Elastic Beanstalk 設定ファイルを使用して、シークレットを環境変数として設定できます。aws:elasticbeanstalk:application:environmentsecrets 名前空間を使用して環境プロパティを定義します。

Example 環境シークレットの .ebextensions/options.config (短縮構文)

```
option_settings:
  aws:elasticbeanstalk:application:environmentsecrets:
    MY_SECRET: arn:aws:secretsmanager:us-east-1:111122223333:secret:mysecret
    MY_PARAMETER: arn:aws:ssm:us-east-1:111122223333:parameter/myparam
```

Example 環境シークレットの .ebextensions/options.config (標準構文)

```
option_settings:
  - namespace: aws:elasticbeanstalk:application:environmentsecrets
    option_name: MY_SECRET
    value: arn:aws:secretsmanager:us-east-1:111122223333:secret:mysecret
  - namespace: aws:elasticbeanstalk:application:environmentsecrets
    option_name: MY_PARAMETER
    value: arn:aws:ssm:us-east-1:111122223333:parameter/myparam
```

を使用した設定 AWS CLI

AWS コマンドラインインターフェイス (AWS CLI) を使用して、シークレットを Elastic Beanstalk 環境変数として設定できます。このセクションでは、aws:elasticbeanstalk:application:environmentsecrets名前空間を使用した create-environment コマンドと update-environment コマンドの例を示します。Elastic Beanstalk がこれらのコマンドが参照する環境の EC2 インスタンスをブートストラップすると、取得されたシークレットとパラメータ値を使用して環境変数が初期化されます。これらの値は、Secrets Manager および Systems Manager パラメータストアのそれぞれの ARNs から取得されます。

次の 2 つの例では、create-environment コマンドを使用して、 および という名前の環境変数として設定されたシークレットMY_SECRETとパラメータを追加しますMY_PARAMETER。

Example 環境変数として設定されたシークレットを持つ create-environment の (名前空間オプションをインラインで)

```
aws elasticbeanstalk create-environment \
```

```
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 20" \
--option-settings \
Namespace=aws:elasticbeanstalk:application:environmentsecrets,OptionName=MY_SECRET,Value=arn:aw
east-1:111122223333:secret:mysecret \
Namespace=aws:elasticbeanstalk:application:environmentsecrets,OptionName=MY_PARAMETER,Value=arn
east-1:111122223333:parameter/myparam
```

別の方法として、options.jsonファイルを使用して、インラインで名前空間オプションを含めるのではなく、名前空間オプションを指定します。

Example 環境変数として設定されたシークレットを持つ create-environment の（ **options.json** ファイルの名前空間オプション）

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 20" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace": "aws:elasticbeanstalk:application:environmentsecrets",
    "OptionName": "MY_SECRET",
    "Value": "arn:aws:secretsmanager:us-east-1:111122223333:secret:mysecret"
  },
  {
    "Namespace": "aws:elasticbeanstalk:application:environmentsecrets",
    "OptionName": "MY_PARAMETER",
    "Value": "arn:aws:ssm:us-east-1:111122223333:parameter/myparam"
  }
]
```

次の例では、 MY_SECRETおよび という名前の環境変数を設定してMY_PARAMETER、既存の環境のシークレットとパラメータを保存します。update-environment コマンドは、create-environmentコマンドと同じ構文のオプションをインラインまたは options.json ファイルで渡します。次の例は、前の例で使用したものと同じoptions.jsonファイルを使用した コマンドを示しています。

Example 環境変数として設定されたシークレットを持つ update-environment の (**options.json** ファイルの名前空間オプション)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 20" \
--option-settings file://options.json
```

AWS SDK を使用した設定

AWS SDKs を使用して、シークレットとパラメータを環境変数として設定できます。前のセクションで説明した update-environmentおよび create-environment AWS CLI コマンドと同様に、CreateEnvironment および UpdateEnvironment API アクションを使用できます。OptionSettings リクエストパラメータを使用して、aws:elasticbeanstalk:application:environmentsecrets名前空間のオプションを指定します。

Elastic Beanstalk 環境変数とのシークレット同期のベストプラクティス

このトピックでは、アプリケーションが Secrets Manager または Systems Manager パラメータストアで環境シークレットを使用するためのベストプラクティスを推奨します。シークレットストアデータが更新またはローテーションされた場合、Elastic Beanstalk アプリケーションは更新された値を自動的に受信しません。Elastic Beanstalk は、インスタンスのブートストラップ時にのみシークレットを環境変数にプルします。

環境変数の更新

Elastic Beanstalk 環境をトリガーしてシークレットストアからシークレットの最新の値を再取得するには、 UpdateEnvironment または RestartAppServerオペレーションを実行することをお勧めします。これらのオペレーションは、Elastic Beanstalk コンソール、 AWS CLI、または Elastic Beanstalk API を使用して実行できます。詳細については、AWS CLI 「Elastic Beanstalk の例」またはAWS Elastic Beanstalk 「 API リファレンス」を参照してください。

シークレット同期に対する自動スケーリング効果の管理

シークレットストアの更新後にスケールアウトイベントまたはインスタンスの置き換えが発生した場合、新しいインスタンスには Secrets Manager または Systems Manager パラメータストアの最新のシークレット値が含まれます。このようなイベントは、環境内の他のすべてのインスタンスが新しいシークレットを取得するために更新されていない場合でも発生する可能性があります。

> **Important**
>
> アプリケーションが同じ環境変数に対して 2 つの異なるシークレット値を使用できるようにする必要があります。これにより、Secrets Manager または Systems Manager パラメータストアでシークレットの更新が発生し、その後に環境でスケールアウトまたはインスタンスの置き換えが行われるイベントに対応できます。他のインスタンスは環境変数の更新を保留中です。更新の待機期間中、すべての環境インスタンスがシークレットストア環境変数に対して同じ値を持つわけではありません。

このようなユースケースの例は、データベース認証情報のローテーションです。スケールアウトイベントが認証情報のローテーションに続く場合、新しくブートストラップされたインスタンスによって参照される環境シークレットには、更新されたデータベース認証情報が含まれます。ただし、既存のインスタンスによって参照される環境シークレットは、 `UpdateEnvironment`または`RestartAppServer`オペレーションによって更新されるまで古い値を保持します。

Amazon Linux 2 環境変数の複数行の値

複数行の値は複数の行で構成され、改行文字が含まれます。Docker および ECS ベースの Docker プラットフォームを除き、Amazon Linux 2 で実行されるプラットフォームは、環境変数の複数行値をサポートしていません。

> **Note**
>
> マルチライン値を検出すると、Elastic Beanstalk は影響を受ける環境のデプロイに失敗します。

次のオプションは、複数行の問題の回避策または解決策として役立ちます。

- Amazon Linux 2 環境を Amazon Linux 2023 にアップグレードします。詳細については、「Amazon Linux 2 から Amazon Linux 2023 への移行」を参照してください。

- シークレット値から改行文字を削除します。方法の 1 つは、値をシークレットストアに保存する前に Base64 でエンコードすることです。その後、アプリケーションは環境シークレット変数から参照するときに、値を元の形式にデコードする必要があります。
- Secrets Manager または Systems Manager パラメータストアから直接データを取得するようにアプリケーションコードを設計します。詳細については、「 でのシークレットの取得Secrets Manager の使用」または「パラメータの取得」を参照してくださいSystems Manager パラメータストアの使用。

Elastic Beanstalk がシークレットとパラメータにアクセスするために必要な IAM アクセス許可

AWS Secrets Manager および Parameter Store のシークレットとパラメータを取得するために必要なアクセス許可を環境の EC2 AWS Systems Manager インスタンスに付与する必要があります。アクセス許可は、EC2 インスタンスプロファイルロールを介して EC2 インスタンスに提供されます。???

以下のセクションでは、使用するサービスに応じて、EC2 インスタンスプロファイルに追加する必要がある特定のアクセス許可を一覧表示します。「IAM ユーザーガイド」の「ロールのアクセス許可ポリシーを更新する」に記載されている手順に従って、これらのアクセス許可を追加します。

> **ECS マネージド Docker プラットフォームの IAM アクセス許可**
>
> ECS マネージド Docker プラットフォームには、このトピックで提供されているアクセス許可に追加の IAM アクセス許可が必要です。Elastic Beanstalk 環境変数とシークレットの統合をサポートするために ECS マネージド Docker プラットフォーム環境に必要なすべてのアクセス許可の詳細については、「」を参照してください実行ロールの ARN 形式。

トピック

- Secrets Manager に必要な IAM アクセス許可
- Systems Manager パラメータストアに必要な IAM アクセス許可

Secrets Manager に必要な IAM アクセス許可

次のアクセス許可は、 AWS Secrets Manager ストアから暗号化されたシークレットを取得するアクセス許可を付与します。

- secretsmanager:GetSecretValue
- kms:Decrypt

を復号するアクセス許可 AWS KMS key は、シークレットがデフォルトキーではなくカスタマーマネージドキーを使用する場合にのみ必要です。カスタムキー ARN を追加すると、カスタマーマネージドキーを復号するアクセス許可が追加されます。

Example Secrets Manager と KMS キーのアクセス許可を持つ ポリシー

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"
            ]
        }
    ]
}
```

Systems Manager パラメータストアに必要な IAM アクセス許可

次のアクセス許可は、 パラメータストアから暗号化された AWS Systems Manager パラメータを取得するアクセス許可を付与します。

- ssm:GetParameter
- kms:Decrypt

を復号するアクセス許可 AWS KMS key は、デフォルトキーではなくカスタマーマネージドキーを使用する`SecureString`パラメータタイプでのみ必要です。カスタムキー ARN を追加すると、カス

タマーマネージドキーを復号するアクセス許可が追加されます。暗号化されていない通常のパラメータタイプである `String`および にはStringList、 は必要ありません AWS KMS key。

Example Systems Manager と AWS KMS キーアクセス許可を持つ ポリシー

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:ssm:us-east-1:111122223333:parameter/my-parameter",
                "arn:aws:kms:us-east-1:111122223333:key/my-key"
            ]
        }
    ]
}
```

AWS Secrets Manager および AWS Systems Manager パラメータストアの使用

このトピックでは、コンソールオプションとプログラムオプションの両方を使用したシークレットの作成と取得の詳細について、 と AWS Secrets Manager AWS Systems Manager Parameter Store の概要、料金情報、リファレンスについて説明します。

Secrets Manager について

AWS Secrets Manager は、シークレットをライフサイクル全体で管理、取得、ローテーションするのに役立ちます。管理できるシークレットデータの例としては、データベース認証情報、アプリケーション認証情報、OAuth トークン、API キーなどがあります。Secrets Manager では、シークレットの自動ローテーションスケジュールを設定できます。

Systems Manager パラメータストアについて

Parameter Store は のツールです AWS Systems Manager。設定データ管理と機密管理のための安全な階層型ストレージを提供します。重要な設定データをパラメータ値として管理できます。Parameter Store で管理できるデータの例には、Amazon マシンイメージ (AMI) IDs、ライセンスコード、パスワード、データベース文字列などがあります。

> **料金**
>
> Secrets Manager と Systems Manager パラメータストアの使用には、標準料金が適用されます。料金の詳細については、以下のウェブサイトを参照してください。
>
> - AWS Secrets Manager 料金
> - AWS Systems Manager 料金 (コンテンツリストから Parameter Store を選択)

トピック

- Secrets Manager を使用してシークレットを作成および取得する
- Systems Manager パラメータストアを使用してパラメータを作成および取得する

Secrets Manager を使用してシークレットを作成および取得する

AWS Secrets Manager コンソール、、または AWS SDK を使用して Secrets Manager シークレットを作成 AWS CLIおよび取得できます。Secrets Manager シークレットを作成および取得するさまざまな方法の詳細については、次のリソースを参照してください。

シークレットの作成

- コンソール – ユーザーガイドの「 AWS Secrets Manager シークレットの作成 (コンソール) AWS Secrets Manager 」
- AWS CLI – AWS Secrets Manager ユーザーガイドの AWS CLI
- AWS SDK – AWS Secrets Manager ユーザーガイドの AWS SDK

シークレットの取得

- コンソール – ユーザーガイドのシークレット値を取得する (コンソール) AWS Secrets Manager
- AWS CLI – ユーザーガイドのシークレット値 (AWS CLI) を取得する AWS Secrets Manager
- AWS SDK – SDK コード例コードライブラリの AWS SDKs を使用した Secrets Manager のコード例 AWS

- その他の方法 – ユーザーガイドの「 からシークレット AWS Secrets Managerを取得するAWS Secrets Manager 」

詳細については AWS Secrets Manager、「 AWS Secrets Manager ユーザーガイド」の「What is AWS Secrets Manager ? 」を参照してください。

Systems Manager パラメータストアを使用してパラメータを作成および取得する

AWS Systems Manager コンソール、、または AWS SDK を使用して AWS CLI、Parameter Store パラメータを作成および取得できます。Parameter Store パラメータを作成および取得するさまざまな方法の詳細については、以下のリソースを参照してください。

パラメータの作成

- コンソール – ユーザーガイドの Systems Manager パラメータを作成する (コンソール) AWS Systems Manager
- AWS CLI – ユーザーガイドの Systems Manager パラメータ (AWS CLI) を作成する AWS Systems Manager
- AWS SDK – SDK または AWS SDK コード例コードライブラリで PutParameter を使用する AWS CLI AWS

パラメータの取得

- コンソール – ユーザーガイドのパラメータ (コンソール) の検索 AWS Systems Manager
- AWS CLI – SDK または AWS SDK コード例コードライブラリで GetParameter を使用する AWS CLI AWS
- AWS SDK – SDK または AWS SDK コード例コードライブラリで GetParameter を使用する AWS CLI AWS

詳細については「AWS Systems Manager ユーザーガイド」の「AWS Systems Manager パラメータストア」を参照してください。

シークレットと Elastic Beanstalk 環境変数の統合のトラブルシューティング

イベント： インスタンスのデプロイで 1 つ以上のシークレットを取得できませんでした

このメッセージは、Elastic Beanstalk がアプリケーションのデプロイ中に指定された 1 つ以上のシークレットを取得できなかったことを示します。

- 環境変数設定の ARN 値で指定されたリソースが存在することを確認します。
- Elastic Beanstalk EC2 インスタンスプロファイルロールに、リソースにアクセスするために必要な IAM アクセス許可があることを確認します。
- このイベントが `RestartAppServer`オペレーションによってトリガーされた場合、問題が修正されたら、`RestartAppServer`呼び出しを再試行して問題を解決します。
- イベントが `UpdateEnvironment`呼び出しによってトリガーされた場合は、`UpdateEnvironment`オペレーションを再試行します。

これらのコマンドの例については、AWS CLI 「Elastic Beanstalk の例」を参照してください。これらのオペレーションの API アクションの詳細については、AWS Elastic Beanstalk 「 API リファレンス」を参照してください。

イベント： インスタンスのデプロイで、このプラットフォームではサポートされていない 1 つ以上の複数行の環境値が検出されました

複数行変数は、Docker および ECS マネージド Docker プラットフォームを除き、Amazon Linux 2 プラットフォームではサポートされていません。続行できるオプションについては、「」を参照してください複数行の値。

イベント： シークレットが指定されると CreateEnvironment が失敗する

が`CreateEnvironment`失敗し、環境変数としてシークレットがある場合は、根本的な問題に対処してから、 `UpdateEnvironment` を使用して環境設定を完了する必要があります。このような状況では環境を起動するには十分ではないため`RestartAppServer`、 を使用しないでください。これらのコマンドの例については、AWS CLI 「Elastic Beanstalk の例」を参照してください。これらのオペレーションの API アクションの詳細については、AWS Elastic Beanstalk 「 API リファレンス」を参照してください。

Amazon VPC で Elastic Beanstalk を使用する

このトピックでは、Elastic Beanstalk で VPC エンドポイントを使用する利点と、実装できるさまざまなタイプの設定について説明します。

Amazon Virtual Private Cloud (Amazon VPC) を使用して、Elastic Beanstalk アプリケーションおよび関連 AWS リソース用の安全なネットワークを作成できます。環境作成時に、アプリケーションイ

ンスタンスとロードバランサーに使用する VPC、サブネットおよびセキュリティグループを選択します。以下の要件を満たしている限り、どの VPC 設定でも使用できます。

VPC の要件

- インターネットアクセス – インスタンスは、次のいずれかのメソッドを介したインターネットアクセスを持つことができます。
  - パブリックサブネット – インスタンスにはパブリック IP アドレスがあり、インターネットゲートウェイを使用してインターネットにアクセスします。
  - プライベートサブネット – インスタンスは NAT デバイスを使用してインターネットにアクセスします。

> **Note**
>
> `elasticbeanstalk` と `elasticbeanstalk-health` のサービスの両方に接続するように VPC の VPC エンドポイント を設定する場合、インターネットアクセスはオプションであり、アプリケーションで特に必要な場合にのみ要求されます。VPC エンドポイントがない場合、VPC でインターネットにアクセスできる必要があります。
> Elastic Beanstalk がセットアップするデフォルトの VPC では、インターネットアクセスが提供されます。

  Elastic Beanstalk は、ウェブプロキシの設定に `HTTPS_PROXY` のようなプロキシ設定をサポートしていません。

- NTP – Elastic Beanstalk 環境内のインスタンスでは、Network Time Protocol (NTP) を使用してシステムクロックを同期させます。インスタンスが UDP ポート 123 上で通信できない場合は、クロックが同期しなくなり、Elastic Beanstalk ヘルスレポートに問題が発生することがあります。これらの問題を回避するには、VPC のセキュリティグループとネットワーク ACL で、ポート 123 上のインバウンドとアウトバウンドの UDP トラフィックを許可していることを確認してください。

elastic-beanstalk-samples リポジトリには、Elastic Beanstalk 環境で使用するために VPC を作成するために使用できる AWS CloudFormation テンプレートが用意されています。

AWS CloudFormation テンプレートを使用してリソースを作成するには

1. サンプルリポジトリをクローンするか、または README にあるリンクを使用してテンプレートをダウンロードします。
2. AWS CloudFormation コンソールを開きます。
3. [スタックの作成] を選択してください。
4. テンプレートを Amazon S3 にアップロード を選択します。
5. ファイルのアップロード を選択し、ローカルマシンからテンプレートファイルをアップロードします。
6. 次へ を選択し、手順どおりにテンプレート内のリソースでスタックを作成します。

スタックの作成が完了したら、[出力] タブを確認して VPC ID およびサブネット ID を探します。これらを使用して、新規の環境ウィザードネットワーク設定カテゴリで VPC を設定します。

トピック

- パブリック VPC
- パブリック/プライベート VPC
- プライベート VPC
- 例: 拠点ホストを持つ VPC で Elastic Beanstalk アプリケーションを起動する
- 例: Amazon RDS を使用して VPC で Elastic Beanstalk を起動する
- VPC エンドポイントでの Elastic Beanstalk の使用
- 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

パブリック VPC

AWS CloudFormation テンプレート – vpc-public.yaml

設定 (ロードバランサー)

- ロードバランサーの可視性 – 公開
- ロードバランサーのサブネット – どちらもパブリックサブネット
- インスタンスのパブリック IP – 有効
- インスタンスのサブネット – どちらもパブリックサブネット
- インスタンスのセキュリティグループ – デフォルトのセキュリティグループを追加

設定 (単一インスタンス)

- インスタンスのサブネット – どちらかのパブリックサブネット
- インスタンスのセキュリティグループ – デフォルトのセキュリティグループを追加

基本的な パブリックのみ の VPC レイアウトには、1 つ以上のパブリックサブネット、1 つのインターネットゲートウェイ、VPC 内のリソース間でのトラフィックを許可するデフォルトの 1 つのセキュリティグループが含まれています。VPC で環境を作成すると、Elastic Beanstalk は環境タイプによって異なる追加のリソースを作成します。

VPC リソース

- 単一インスタンス – Elastic Beanstalk は、インターネットからポート 80 へのトラフィックを許可するセキュリティグループをアプリケーションインスタンスに作成し、Elastic IP をそのインスタンスに割り当てて、パブリック IP アドレスを付与します。環境のドメイン名はインスタンスのパブリック IP アドレスを解決します。
- 負荷分散 – Elastic Beanstalk は、インターネットからポート 80 へのトラフィックを許可するセキュリティグループ、およびロードバランサーのセキュリティグループからのトラフィックを許可するセキュリティグループをアプリケーションインスタンスに作成します。環境のドメイン名は、ロードバランサーのパブリックドメイン名を解決します。

これは、デフォルト VPC を使用する場合に Elastic Beanstalk がネットワークを管理する方法と類似しています。パブリックサブネットのセキュリティは、Elastic Beanstalk が作成するロードバランサーとインスタンスのセキュリティグループに依存します。これには NAT ゲートウェイが必要ではないため、最もコストが低い設定です。

パブリック/プライベート VPC

AWS CloudFormation テンプレート – vpc-privatepublic.yaml

設定 (ロードバランサー)

- ロードバランサーの可視性 – 公開
- ロードバランサーのサブネット – どちらもパブリックサブネット
- インスタンスのパブリック IP – 無効
- インスタンスのサブネット – どちらもプライベートサブネット
- インスタンスのセキュリティグループ – デフォルトのセキュリティグループを追加

セキュリティを高めるには、追加のプライベートサブネットを VPC に追加して、パブリック/プライベートレイアウトを作成します。このレイアウトではパブリックサブネットのロードバランサーおよび NAT ゲートウェイが必要であり、プライベートサブネットでアプリケーションインスタンス、データベースや他のリソースを実行することができるようになります。プライベートサブネットのインスタンスは、ロードバランサーおよび NAT ゲートウェイを介したインターネットのみと通信できます。

プライベート VPC

AWS CloudFormation テンプレート – vpc-private.yaml

設定 (ロードバランサー)

- ロードバランサーの可視性 – プライベート
- ロードバランサーサブネット – 両方のプライベートサブネット
- インスタンスのパブリック IP – 無効
- インスタンスのサブネット – どちらもプライベートサブネット
- インスタンスのセキュリティグループ – デフォルトのセキュリティグループを追加

インターネットからのアクセスを希望しない内部アプリケーションについては、すべてをプライベートサブネットで実行し、ロードバランサーを内部用に設定します ([ロードバランサーの可視性] を [内部] に変更します)。このテンプレートは、パブリックサブネットとインターネットゲートウェイを持たない VPC を作成します。このレイアウトは、同じ VPC あるいはアタッチされた VPN からのみアクセスできる必要があるアプリケーションに使用します。

プライベート VPC で Elastic Beanstalk 環境を実行する

プライベート VPC で Elastic Beanstalk 環境を作成する場合、その環境ではインターネットにアクセスできません。アプリケーションは、Elastic Beanstalk サービスまたはその他のサービスへのアクセスを必要とする場合があります。環境で拡張ヘルスレポートを使用する場合があります。この場合、環境インスタンスによって拡張ヘルスサービスにヘルス情報が送信されます。また、環境インスタンスの Elastic Beanstalk コードは、トラフィックを他の AWS サービスに送信し、その他のトラフィックをAWS エンドポイント以外のポイントに送信します (たとえば、アプリケーションの依存関係パッケージをダウンロードする場合など) 。ここでは、環境が正しく動作するようにするため、この場合に実行する必要がある手順をいくつか示します。

- Elastic Beanstalk の VPC エンドポイントを設定する – Elastic Beanstalk および拡張ヘルスサービスは、VPC エンドポイントをサポートします。これにより、これらのサービスへのトラフィック

は Amazon ネットワーク内に留まり、インターネットアクセスが不要になります。詳細については、「the section called “VPC エンドポイント”」を参照してください。

- 追加サービス用に VPC エンドポイントを設定する – Elastic Beanstalk インスタンスは、 AWS ユーザーに代わって Amazon Simple Storage Service (Amazon S3)、Amazon Simple Queue Service (Amazon SQS) AWS CloudFormation、Amazon CloudWatch Logs などの他のサービスにトラフィックを送信します。これらのサービスの VPC エンドポイントも設定する必要があります。サービスごとのリンクなど、VPC エンドポイントの詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイント」を参照してください。

> **Note**
>
> Elastic Beanstalk を含む一部の AWS サービスは、限られた数の AWS リージョンで VPC エンドポイントをサポートしています。プライベート VPC ソリューションを設計するときは、ここで説明する Elastic Beanstalk およびその他の依存サービスが、選択した AWS リージョンの VPC エンドポイントをサポートしていることを確認します。

- プライベート Docker イメージを提供する – Docker 環境では、環境のインスタンスのコードが、環境の作成中にインターネットから設定済みの Docker イメージを取得しようとして失敗することがあります。この問題を回避するには、環境にカスタム Docker イメージをビルドするか、Amazon Elastic Container Registry (Amazon ECR) に保存された Docker イメージを使用して、Amazon ECR サービスの VPC エンドポイントを設定します。
- DNS 名を有効にする – 環境インスタンスの Elastic Beanstalk コードは、パブリックエンドポイントを使用してすべての AWS サービスにトラフィックを送信します。このトラフィックを確実に通過させるには、すべてのインターフェイス VPC エンドポイントを設定する際に、[Enable DNS name (DNS 名を有効にする)] オプションを選択します。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。

> **Important**
>
> VPC がプライベートではなく、パブリックインターネットにアクセスできる場合、および VPC エンドポイントで [Enable DNS name (DNS 名を有効にする)] が無効になっている場合は、それぞれのサービスへのトラフィックはパブリックインターネットを経由します。これは意図した動作ではない場合があります。プライベート VPC でこの問題を検出するのは簡単です。トラフィックが通過するのを防ぎ、エラーを受信できるためです。ただし、パブリック向け VPC では何も表示されません。

- アプリケーションの依存関係を含める – アプリケーションに言語ランタイムパッケージなどの依存関係がある場合、環境の作成中にインターネットからダウンロードおよびインストールしようとして失敗する可能性があります。このエラーを回避するには、アプリケーションのソースバンドルにすべての依存関係パッケージを含めます。
- 現在のプラットフォームバージョンを使用する – ご使用の環境で、2020 年 2 月 24 日以降にリリースされたプラットフォームバージョンが使用されていることを確認してください。具体的には、Linux 更新プログラム 2020-02-28、Windows 更新プログラム 2020-02-24 の 2 つの更新プログラムのいずれか後にリリースされたプラットフォームのバージョンを使用します。

> **Note**
>
> プラットフォームのバージョンを更新する必要がある理由は、古いバージョンに [Enable DNS name (DNS 名を有効にする)] オプションで作成された DNS エントリが Amazon SQS で正しく機能しないという問題があったためです。

例: 拠点ホストを持つ VPC で Elastic Beanstalk アプリケーションを起動する

このセクションでは、踏み台ホストを使用して VPC 内に Elastic Beanstalk アプリケーションをデプロイする方法と、このトポロジを実装する理由について説明します。

プライベートサブネット内に配置されている Amazon EC2 インスタンスに、リモートで接続することはできません。インスタンスに接続するには、プロキシとして動作する拠点サーバーをパブリックサブネットに設定できます。例えば、パブリックサブネットで SSH ポートフォーワード機能または RDP ゲートウェイを設定し、ご自身のネットワークからデータベースサーバーに向かうトラフィックをプロキシできます。このセクションの例では、プライベートサブネットおよびパブリックサブネットで VPC を作成する方法を示しています。インスタンスはプライベートサブネット内に、拠点ホスト、NAT ゲートウェイ、およびロードバランサーはパブリックサブネット内に配置されています。インフラストラクチャは次の図と似ています。

拠点ホストを使用して VPC 内に Elastic Beanstalk アプリケーションをデプロイするには、以下のサブセクションで説明するステップを実行します。

ステップ

- パブリックサブネットとプライベートサブネットでの VPC の作成

- 拠点ホストのセキュリティグループを作成および設定する
- インスタンスのセキュリティグループの更新
- 拠点ホストの作成

パブリックサブネットとプライベートサブネットでの VPC の作成

「パブリック/プライベート VPC」のすべての手順を完了します。アプリケーションをデプロイする際、そのインスタンスの Amazon EC2 キーペアを指定して、リモートで接続できるようにする必要があります。インスタンスのキーペアを指定する方法の詳細については、「お客様のElastic Beanstalk 環境に対する Amazon EC2 インスタンス」を参照してください。

拠点ホストのセキュリティグループを作成および設定する

拠点ホストのセキュリティグループを作成し、インターネットからのインバウンド SSH トラフィックと Amazon EC2 インスタンスが含まれるプライベートサブネットへのアウトバウンド SSH トラフィックを許可するルールを追加します。

拠点ホストのセキュリティグループを作成するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. ナビゲーションペインで、セキュリティグループ] を選択します。
3. [Create Security Group (セキュリティグループの作成)] を選択します。
4. [セキュリティグループの作成] ダイアログボックスで、次の内容を入力して [はい、作成する] を選択します。

   名前タグ (オプション)

   セキュリティグループの名前タグを入力します。

   グループ名

   セキュリティグループの名前を入力します。

   説明

   セキュリティグループの説明を入力します。

   [VPC]

   VPC を選択します。

セキュリティグループが作成され、[セキュリティグループ] ページに表示されます。ID (sg-xxxxxxxx など) が付いている点に注目してください。ページの右上にある [表示/非表示] をクリックして、[グループ ID] の列をオンにする必要がある場合があります。

踏み台ホストのセキュリティグループを設定するには

1. セキュリティグループのリストで、拠点ホストに対して作成したばかりのセキュリティグループのチェックボックスを選択します。
2. [Inbound Rules] タブで、[Edit] を選択します。
3. 必要に応じて、[別のルールの追加] を選択します。
4. 拠点ホストが Linux インスタンスである場合は、[タイプ] で [SSH] を選択します。

   拠点ホストが Windows インスタンスである場合は、[タイプ] で [RDP] を選択します。
5. [ソース] フィールドに希望するソース CIDR の範囲を入力して、[保存] を選択します。

6. [アウトバウンドルール] タブで [編集] を選択します。
7. 必要に応じて、[別のルールの追加] を選択します。
8. [タイプ] で、インバウンドルールに指定したタイプを選択します。
9. [ソース] フィールドに、VPC のプライベートサブネットにおけるホストのサブネットの CIDR 範囲を入力します。

   検索するには:

   a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
   b. ナビゲーションペインで、[サブネット] を選択してください。
   c. 踏み台ホストのブリッジ先のホストがあるアベイラビリティーゾーンごとに [IPv4 CIDR] の値を書き留めます。

   > **Note**
   > 複数のアベイラビリティーゾーンにホストがある場合は、これらのアベイラビリティーゾーンのそれぞれに対してアウトバウンドルールを作成します。

10. [Save] を選択します。

インスタンスのセキュリティグループの更新

デフォルトでは、インスタンス用に作成したセキュリティグループでは着信トラフィックを許可していません。Elastic Beanstalk はインスタンスのデフォルトグループを変更して SSH トラフィックを許可しますが、使用するインスタンスが Windows インスタンスの場合は、セキュリティグループで RDP トラフィックを許可するよう、カスタムインスタンスを変更する必要があります。

RDP のインスタンスセキュリティグループを更新するには

1. セキュリティグループのリストで、インスタンスセキュリティグループのチェックボックスを選択します。
2. [インバウンド] タブで、[編集] を選択します。
3. 必要に応じて、[別のルールの追加] を選択します。
4. 以下の値を入力して、[保存] を選択します。

   タイプ

   `RDP`

   プロトコル

   `TCP`

   ポート範囲

   `3389`

   ソース

   拠点ホストセキュリティグループの ID（`sg-8a6f71e8` など）を入力し、[保存] を選択します。

拠点ホストの作成

拠点ホストを作成するには、拠点ホストとして動作するパブリックサブネットで Amazon EC2 インスタンスを起動します。

プライベートサブネットで Windows インスタンスの拠点ホストをセットアップする詳細については、「踏み台サーバーを使用した EC2 インスタンスへのネットワークアクセスの制御」を参照してください。

プライベートサブネットで Linux インスタンスの拠点ホストをセットアップする詳細については、「プライベート Amazon VPC で実行中の Linux インスタンスに安全に接続します」を参照してください。

例: Amazon RDS を使用して VPC で Elastic Beanstalk を起動する

このセクションでは、NAT ゲートウェイを使用して VPC で Amazon RDS によって Elastic Beanstalk アプリケーションをデプロイするタスクについて説明します。

インフラストラクチャは次の図と似ています。

> **Note**
>
> これまでアプリケーションで DB インスタンスを使用したことがない場合は、テスト環境に DB インスタンスを追加し、両方に VPC 設定を追加する前に、外部 DB インスタンスへの接続を試みます。

パブリックサブネットとプライベートサブネットでの VPC の作成

Amazon VPC コンソールを使用して VPC を作成します。

VPC を作成するには

1. Amazon VPC コンソールにサインインします。
2. ナビゲーションペインで、[VPC ダッシュボード] を選択します。続いて、[VPC の作成] を選択します。
3. [VPC with Public and Private Subnets(パブリックサブネットとプライベートサブネットを持つ VPC)]、[選択] の順に選択します。

4. Elastic Load Balancing ロードバランサーと Amazon EC2 インスタンスが通信できるようにするには、これらが同じアベイラビリティーゾーンにある必要があります。各 [アベイラビリティーゾーン] リストから同じアベイラビリティーゾーンを選択します。

5. NAT ゲートウェイの Elastic IP アドレスを選択します。
6. [Create VPC (VPC の作成)] を選択します。

   ウィザードが、VPC、サブネット、インターネットゲートウェイの作成を開始します。また、メインルートテーブルを更新して、カスタムルートテーブルを作成します。最後に、ウィザードはパブリックサブネットで NAT ゲートウェイを作成します。

   > **Note**
   >
   > NAT ゲートウェイの代わりにパブリックサブネットで NATインスタンスを起動することを選択できます。詳細については、「Amazon VPC ユーザーガイド」の「シナリオ 2: パブリックサブネットとプライベートサブネットを使用する VPC (NAT)」を参照してください。

7. VPC が正しく作成されると、VPC ID が発行されます。この値は次のステップで必要になります。VPC ID を表示するには、Amazon VPC コンソールの左ペインで [VPC] を選択します。

DB サブネットグループを作成する

VPC の DB サブネットグループは、バックエンド RDS DB インスタンス用に指定できるサブネットのコレクション (通常はプライベート) です。各 DB サブネットグループには、特定の AWS リージョンの各アベイラビリティーゾーンに少なくとも 1 つのサブネットが必要です。詳細については、「VPC でサブネットを作成する」を参照してください。

DB サブネットグループを作成する

1. Amazon RDS コンソールを開きます。
2. [Navigation] ペインで、[Subnet groups] を選択します。
3. [Create DB Subnet Group] を選択します。
4. [名前] を選択し、DB サブネットグループの名前を入力します。
5. [Description (説明)] を選択して、DB サブネットグループの説明を入力します。
6. [VPC] で、作成した VPC の ID を選択します。
7. [Add subnets (サブネットの追加)] で、[Add all the subnets related to this VPC (この VPC に関連するすべてのサブネットを追加)] を選択します。

8. 完了したら、[Create] を選択します。

   Amazon RDS コンソールの DB サブネットグループのリストに新しい DB サブネットグループが表示されます。その DB サブネットグループを選択すると、そのグループに関連付けられているすべてのサブネットなどの詳細情報が、ページの下部にある詳細ペインに表示されます。

Elastic Beanstalk にデプロイします

VPC を設定したら、VPC 内部に環境を作成して、アプリケーションを Elastic Beanstalk にデプロイできます。これを行うには、Elastic Beanstalk コンソールを使用するか、ツールキット、EB CLI AWS CLI、または Elastic Beanstalk API を使用できます AWS 。Elastic Beanstalk コンソールを使用した場合、`.war` または `.zip` ファイルをアップロードして、ウィザード内で VPC 設定を選択するだけでかまいません。その後、Elastic Beanstalk は VPC 内に環境を作成し、アプリケーションをデプロイします。または、 AWS ツールキット AWS CLI、EB CLI、または Elastic Beanstalk API を使用してアプリケーションをデプロイすることもできます。その場合は、設定ファイルで VPC オプション設定を定義して、このファイルをソースバンドルと共にデプロイする必要があります。このトピックでは、両方の方法の手順を説明します。

Elastic Beanstalk コンソールでデプロイする

Elastic Beanstalk コンソールに、VPC 内部に新しい環境を作成する手順が表示されます。`.war` ファイル (Java アプリケーション)、または `.zip` ファイル (その他すべてのアプリケーション) を提供する必要があります。Elastic Beanstalk 環境ウィザードの [VPC 設定] ページで、次の選択を行う必要があります。

[VPC]

VPC を選択します。

VPC セキュリティグループ

上の手順で作成したインスタンスセキュリティグループを選択します。

ELB の可視性

ロードバランサーを公開する場合は `External` を選択し、ロードバランサーを VPC 内でのみ使用できるようにする場合は `Internal` を選択します。

ロードバランサーと EC2 インスタンスのサブネットを選択します。必ず、ロードバランサーのパブリックサブネットと Amazon EC2 インスタンスのプライベートサブネットを選択します。デフォル

トでは、VPC 作成ウィザードにより `10.0.0.0/24` にパブリックサブネットが、`10.0.1.0/24` にプライベートサブネットが作成されます。

サブネット ID を表示するには、Amazon VPC コンソールで [サブネット] を選択します。

AWS ツールキット、EB CLI AWS CLI、または API を使用したデプロイ

AWS ツールキット、EB CLI、または API を使用してアプリケーションを Elastic Beanstalk にデプロイする場合、ファイルで VPC オプション設定を指定し AWS CLI、ソースバンドルでデプロイできます。詳細については「設定ファイル (`.ebextensions`) による高度な環境のカスタマイズ」を参照してください。

オプション設定を更新するときに、少なくとも次を指定する必要があります。

- VPCId – VPC の ID が含まれます。
- Subnets – Auto Scaling グループサブネットの ID が含まれます。この例では、これはプライベートサブネットの ID です。
- ELBSubnets – ロードバランサーのサブネットの ID が含まれます。この例では、これはパブリックサブネットの ID です。
- SecurityGroups – セキュリティグループの ID が含まれます。
- DBSubnets – DB サブネットの ID が含まれます。

> **Note**
>
> DB サブネットを使用する場合は、VPC に追加のサブネットを作成して、 AWS リージョン内のすべてのアベイラビリティーゾーンをカバーする必要があります。

オプションで、次の情報を指定することもできます。

- ELBScheme – VPC 外部から Elastic Beanstalk アプリケーションにアクセスできないようにするために VPC 内に内部ロードバランサーを作成する場合は、`internal` を指定します。

VPC 内で Elastic Beanstalk アプリケーションをデプロイするときに使用可能なオプション設定の例を次に示します。VPC オプション設定の詳細 (オプション設定の指定方法の例、デフォルト値、有効な値など) については、「設定オプション」に記載されている aws:ec2:vpc 名前空間の表を参照してください。

```
option_settings:
  - namespace: aws:autoscaling:launchconfiguration
    option_name: EC2KeyName
    value: ec2keypair

  - namespace: aws:ec2:vpc
    option_name: VPCId
    value: vpc-170647c

  - namespace: aws:ec2:vpc
    option_name: Subnets
    value: subnet-4f195024

  - namespace: aws:ec2:vpc
    option_name: ELBSubnets
    value: subnet-fe064f95

  - namespace: aws:ec2:vpc
    option_name: DBSubnets
    value: subnet-fg148g78

  - namespace: aws:autoscaling:launchconfiguration
    option_name: InstanceType
    value: m1.small

  - namespace: aws:autoscaling:launchconfiguration
    option_name: SecurityGroups
    value: sg-7f1ef110
```

> **Note**
>
> DB サブネットを使用する場合は、VPC に AWS 、リージョン内のすべてのアベイラビリティーゾーンをカバーするサブネットがあることを確認してください。

VPC エンドポイントでの Elastic Beanstalk の使用

このトピックでは、Elastic Beanstalk アプリケーションに対する VPC エンドポイントの利点について説明します。また、Elastic Beanstalk サービスへのインターフェイス VPC エンドポイントを作成する手順も示します。

VPC エンドポイントを使用すると、インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続を必要とせずに AWS PrivateLink、サポートされている AWS のサービスや を搭載した 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。

VPC のインスタンスでは、サービスのリソースと通信するためにパブリック IP アドレスを必要としません。VPC と他のサービス間のトラフィックは、Amazon ネットワークを離れることはありません。VPC エンドポイントの詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイント」を参照してください。

AWS Elastic Beanstalk は AWS PrivateLink、Elastic Beanstalk サービスへのプライベート接続を提供し、パブリックインターネットへのトラフィックの露出を排除する をサポートします。アプリケーションが を使用して Elastic Beanstalk 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 VPC エンドポイント (インターフェイスエンドポイント) と呼ばれるタイプの VPC エンドポイントを設定します。詳細については、「Amazon VPC ユーザーガイド」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。

> **Note**
>
> Elastic Beanstalk は、限られた数の AWS PrivateLink およびインターフェイス VPC エンドポイントをサポートしています AWS リージョン。近い将来、より多くの AWS リージョンにサポートを拡張する予定です。

IPv6 サポート

Elastic Beanstalk は、IPv4 および IPv6 を介した受信トラフィックをサポートします。このセクションでは、IPV6 をサポートするパブリックエンドポイントについて説明し、デュアルスタックトラフィックをサポートするように Elastic Beanstalk VPC エンドポイントを設定する方法についても説明します。

IPv6 に関する一般的な情報については、「Amazon VPC ユーザーガイド」のAWS IPv6 をサポートする のサービス」と、 の AWS ホワイトペーパーIPv6 AWS」を参照してください。

パブリックエンドポイント

Elastic Beanstalk サービスには、古い IPv4 エンドポイントと、それより新しいデュアルスタック機能を備えたエンドポイントで構成される 2 つのエンドポイントセットがあります。両方のエンドポイントセットは AWS 、命名基準に従います。

- IPv4 エンドポイントはドメイン `amazonaws.com` を使用します – 一般的なサービスエンドポイントの形式: `elasticbeanstalk.region.amazonaws.com`
- デュアルスタックエンドポイントはドメイン `api.aws` を使用します – 一般的なサービスエンドポイントの形式: `elasticbeanstalk.region.api.aws`

サービスヘルスと FIPS のエンドポイントのホスト名は異なりますが、同じドメイン名パターンに従います。エンドポイントのリストについては、「Amazon Web Services 全般のリファレンス」の「Elastic Beanstalk サービスエンドポイント」を参照してください。

Elastic Beanstalk へのリクエスト

AWS CLI または AWS SDK を使用して Elastic Beanstalk サービスにリクエストを送信する場合、IPv4 エンドポイントまたはデュアルスタックエンドポイントを指定できます。エンドポイント URL が指定されていない場合、 AWS CLI および AWS SDK はデフォルトで IPv4-onlyエンドポイントを使用します。

次の例は、デュアルスタックのエンドポイントにリクエスト AWS CLI を送信する方法を示しています。

Example

```
aws elasticbeanstalk list-available-solution-stacks \
    --endpoint-url "https://elasticbeanstalk.us-east-1.api.aws"
```

次の例は、デュアルスタックのエンドポイントにリクエストを送信する AWS Python SDK を示しています。

Example

```
import boto3

dual_stack_eb_client = boto3.client(
    service_name='elasticbeanstalk',
    region_name='us-east-1',
    endpoint_url='https://elasticbeanstalk.us-east-1.api.aws';
)

print(dual_stack_eb_client.list_available_solution_stacks())
```

デュアルスタック IP の VPC エンドポイント

デュアルスタックトラフィックをサポートするように Elastic Beanstalk VPC エンドポイントを設定するには、VPC エンドポイントの IP アドレスタイプパラメータにデュアルスタックを指定します。このフィールドは AWS CLI、AWS SDK、または AWS PrivateLink コンソールから指定できます。AWS PrivateLink コンソールでこれを行う手順については、「 AWS PrivateLink ガイド」の「VPC エンドポイントの作成」を参照してください。

> **Note**
>
> VPC エンドポイントの IP アドレスタイプには、IPv4 または デュアルスタックのいずれかを指定する必要があります。現時点では、Elastic Beanstalk VPC エンドポイントは、IPv6 のみのサポートであることを示す IPv6 の IP アドレスタイプをサポートしていません。デュアルスタックオプションでは、IPv4 と IPv6 の両方のインターネットプロトコルを使用できます。

次の例は、 AWS CLIを使用してデュアルスタック VPC エンドポイントを作成する方法を示しています。

Example

```
aws ec2 create-vpc-endpoint \
    --vpc-id "vpc-example"
    --service-name "com.amazonaws.us-east-1.elasticbeanstalk"
    --ip-address-type "dualstack"
```

Elastic Beanstalk 用の VPC エンドポイントをセットアップする

VPC で Elastic Beanstalk 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、「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。

- [サービス名] で、com.amazonaws. ***region***.elasticbeanstalk を選択します。
- [IP アドレスタイプ] で、[IPv4] または [デュアルスタック] を選択します。現時点では、Elastic Beanstalk VPC エンドポイントは、IPv6 のみのサポートであることを示す IPv6 の IP アドレスタイプをサポートしていません。デュアルスタックオプションでは、IPv4 と IPv6 の両方のインターネットプロトコルを使用できます。

VPC がパブリックインターネットアクセスで設定されている場合でも、アプリケーションは `elasticbeanstalk.region.amazonaws.com` または `elasticbeanstalk.region.api.aws`

のいずれかの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 Elastic Beanstalk にアクセスできます。これを防ぐには、エンドポイントの作成時に [Enable DNS name (DNS 名を有効にする)] が有効になっていることを確認します (デフォルトでは true)。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。

拡張ヘルスのための VPC エンドポイントを設定する

環境で拡張ヘルスレポートを有効にした場合は、拡張ヘルス情報の送信 AWS PrivateLink 先も設定できます。拡張ヘルス情報は、環境インスタンスの Elastic Beanstalk コンポーネントである `healthd` デーモンによって、別の Elastic Beanstalk 拡張ヘルスサービスに送信されます。VPC でこの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、「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。

- [サービス名] で、com.amazonaws.***region***.elasticbeanstalk-healthを選択します。
- [IP アドレスタイプ] で、[IPv4] または [デュアルスタック] を選択します。現時点では、Elastic Beanstalk VPC エンドポイントは、IPv6 のみのサポートであることを示す IPv6 の IP アドレスタイプをサポートしていません。デュアルスタックオプションでは、IPv4 と IPv6 の両方のインターネットプロトコルを使用できます。

> **Important**
>
> `healthd` デーモンは、拡張ヘルス情報をパブリックエンドポイント `elasticbeanstalk-health.`*`region`*`.amazonaws.com` または `elasticbeanstalk-health.`*`region`*`.api.aws` に送信します。VPC がパブリックインターネットアクセスで設定されていて、VPC エンドポイントで [Enable DNS name (DNS 名を有効にする)] が無効になっている場合、拡張ヘルス情報はパブリックインターネットを通過します。これは、拡張ヘルスの VPC エンドポイントを設定した際に意図したことではない可能性があります。[Enable DNS name (DNS 名を有効にする)] が有効になっていることを確認します (デフォルトでは true)。

プライベート VPC で VPC エンドポイントを使用する

プライベート VPC、または VPC 内のプライベートサブネットにはパブリックインターネットアクセスはありません。プライベート VPC で Elastic Beanstalk 環境を実行し、インターフェイス VPC エンドポイントを設定してセキュリティを強化できます。この場合、Elastic Beanstalk サービスへの連

絡に加えて、他の理由で環境がインターネットに接続しようとする場合があることに注意してください。プライベート VPC で環境を実行する方法の詳細については、「the section called “プライベート VPC で Elastic Beanstalk 環境を実行する”」を参照してください。

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

このトピックでは、VPC エンドポイントにポリシーをアタッチして、アプリケーション (サービス) と Elastic Beanstalk 環境へのアクセスを制御する方法について説明します。

エンドポイントポリシーは、エンドポイントから指定されたサービスへのアクセスを制御する AWS Identity and Access Management (IAM) リソースポリシーです。エンドポイントポリシーは、エンドポイントに固有のものです。これは、環境が持つ可能性のあるユーザーまたはインスタンスの IAM ポリシーとは別のものであり、上書きしたり置き換えたりすることはありません。

デフォルトでは、VPC エンドポイントは、関連付けられているサービスへのフルアクセスを許可します。エンドポイントを作成または変更するときは、エンドポイントポリシーをアタッチして、サービスに関連付けられた特定のリソースへのアクセスを制御できます。VPC エンドポイントポリシーの作成と使用の詳細については、「AWS PrivateLink ガイド」の「エンドポイントポリシーを使用して VPC エンドポイントへのアクセスを制御する」を参照してください。

> **Note**
>
> 制限のあるエンドポイントポリシーを作成するときは、必要なリソースに特定のアクセス許可を追加する必要がある場合があります。これにより、これらのリソースへのアクセスがエンドポイントポリシーによってブロックされなくなります。これにより、環境は引き続き正常にデプロイされ機能します。

次の例では、VPC エンドポイントを経由して環境を終了するアクセス許可をすべてのユーザーに拒否し、他のすべてのアクションへのフルアクセスを許可します。

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
```

```
        },
        {
            "Action": "elasticbeanstalk:TerminateEnvironment",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

制限付き VPC エンドポイントポリシーに必要な Amazon S3 バケットアクセス許可

VPC エンドポイントポリシーに制限を追加する場合は、環境が引き続き正常にデプロイされ機能するように、特定の Amazon S3 バケットアクセス許可を含める必要があります。このセクションでは、必要な S3 バケットについて説明し、ポリシーの例を示します。

トピック

- 環境プラットフォームを管理するためにアセットを保存する S3 バケット
- が所有する S3 バケット AWS CloudFormation
- ソースコードやその他の項目を保存するためにカスタマーアカウントが所有する S3 バケット
- Docker レジストリ認証をサポートするためにカスタマーアカウントが所有する S3 バケット
- VPC エンドポイントポリシーの更新

環境プラットフォームを管理するためにアセットを保存する S3 バケット

Elastic Beanstalk サービスは、ソリューションスタック (プラットフォームバージョン) に関連付けられたアセットを保存する S3 バケットを所有しています。これらのアセットには、設定ファイル、サンプルアプリケーション、使用可能なインスタンスタイプが含まれます。Elastic Beanstalk が環境を作成および管理すると、対応する各アセットバケットから特定のプラットフォームバージョンの必要な情報を取得します AWS リージョン。

S3 バケット ARN

`arn:aws:s3:::elasticbeanstalk-samples-`*`region`*

Amazon Linux 2 以降

- `arn:aws:s3:::elasticbeanstalk-platform-assets-`*`region`*

> **Note**
> バケット名は、BJS リージョンの別の規則に従います。文字列 public-beta-cn-north-1 が、`#####`の代わりに使用されます。例えば、`arn:aws:s3:::elasticbeanstalk-platform-assets-public-beta-cn-north-1` です。

Windows Server、Amazon Linux (AMI)、Amazon Linux 2 以降

- `arn:aws:s3:::elasticbeanstalk-env-resources-`*`region`*
- `arn:aws:s3:::elasticbeanstalk-`*`region`*

> **Note**
> platform-assets バケットと env-resources バケットのバケット名は、一部のリージョンで異なる規則に従います。詳細については、以下のリージョン固有のバケット命名パターンセクションを参照してください。

リージョン固有のバケット ARN パターン

リージョン	バケットタイプ	バケット ARN パターン
me-central-1	プラットフォームアセット	`arn:aws:s3:::elasticbeanstalk-platform-assets-me-central-1-f08b818c`
	env-resources	`arn:aws:s3:::elasticbeanstalk-env-resources-me-central-1-f08b818c`

オペレーション

GetObject

VPC エンドポイントポリシーの例

次の例は、米国東部 (オハイオ) リージョン (us-east-2) で Elastic Beanstalk の運用に必要な S3 バケットへのアクセス権を付与する方法を示しています。この例では、Amazon Linux プラットフォー

ムと Windows Server プラットフォームの両方のすべてのバケットを一覧表示します。環境のオペレーティングシステムに適用されるバケットのみを含めるようにポリシーを更新します。

ポリシーの例

> **Important**
>
> このポリシーの特定のリージョンの代わりにワイルドカード文字 (*) を使用しないことをお勧めします。例えば、`arn:aws:s3:::cloudformation-waitcondition-us-east-2/*` を使用して、`arn:aws:s3:::cloudformation-waitcondition-*/*` は使用しないでください。ワイルドカードを使用すると、アクセスを付与する S3 バケットへのアクセス権が付与される場合があります。複数のリージョンでポリシーを使用する場合は、各リージョンの最初の `Statement` ブロックを繰り返すことをお勧めします。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsToAWSResources",
            "Effect": "Allow",
            "Principal": {"AWS": "*"},
            "Action": ["s3:GetObject"],
            "Resource": [
                "arn:aws:s3:::elasticbeanstalk-platform-assets-us-east-2/*",
                "arn:aws:s3:::elasticbeanstalk-env-resources-us-east-2/*",
                "arn:aws:s3:::elasticbeanstalk-env-resources-us-east-2/*",
                "arn:aws:s3:::elasticbeanstalk-samples-us-east-2/*"
            ]
         }
    ]
}
```

が所有する S3 バケット AWS CloudFormation

Elastic Beanstalk は AWS CloudFormation を使用して環境のリソースを作成します。CloudFormation は、待機条件に対する応答をモニタリング AWS リージョン するために、それぞれに S3 バケットを所有しています。

Elastic Beanstalk などのサービスは CloudFormation が所有する S3 バケットの 署名付き Amazon S3 URL にリクエストを送信することで、CloudFormation と通信します。CloudFormation は、`cloudformation.amazonaws.com` サービスプリンシパルを使用して署名付き Amazon S3 URL を作成します。

詳細については、「AWS CloudFormation ユーザーガイド」の「CloudFormation VPC エンドポイントに関する考慮事項」を参照してください。署名付き URL の詳細については、「Amazon S3 ユーザーガイド」の署名付き URL に関するページを参照してください。

S3 バケット ARN

- `arn:aws:s3:::cloudformation-waitcondition-`*`region`*

  待機条件を使用する場合、リージョン名にはダッシュが含まれます。例えば、us-west-2 などです。
- `arn:aws:s3:::cloudformation-custom-resource-response-`*`region`*

  カスタムリソースを使用する場合、リージョン名にダッシュは含まれません。例えば、uswest2 です。

オペレーション

GetObject

VPC エンドポイントポリシーの例

次の例は、米国東部 (オハイオ) リージョン (us-east-2) で Elastic Beanstalk の運用に必要な S3 バケットへのアクセス権を付与する方法を示しています。

ポリシーの例

> **Important**
>
> このポリシーの特定のリージョンの代わりにワイルドカード文字 (*) を使用しないことをお勧めします。例えば、`arn:aws:s3:::cloudformation-waitcondition-us-east-2/`

* を使用して、arn:aws:s3:::cloudformation-waitcondition-*/* は使用しないでください。ワイルドカードを使用すると、アクセスを付与する S3 バケットへのアクセス権が付与される場合があります。複数のリージョンでポリシーを使用する場合は、各リージョンの最初の Statement ブロックを繰り返すことをお勧めします。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsToCloudFormation",
            "Effect": "Allow",
            "Principal": {"AWS": "*"},
            "Action": ["s3:GetObject"],
            "Resource": [
                "arn:aws:s3:::cloudformation-waitcondition-us-east-2/*",
                "arn:aws:s3:::cloudformation-custom-resource-response-us-east-2/
*"
            ]
         }
    ]
}
```

ソースコードやその他の項目を保存するためにカスタマーアカウントが所有する S3 バケット

このバケットは、環境を所有する AWS 顧客アカウントによって所有されています。ソースコードやリクエストされたログなど、環境に固有のリソースが保存されます。

S3 バケット ARN

arn:aws:s3:::elasticbeanstalk-*region*-*account-id*

オペレーション

- GetObject
- GetObjectAcl
- PutObject

- PutObjectAcl
- ListBucket

VPC エンドポイントポリシーの例

次の例は、米国東部 (オハイオ) リージョン (us-east-2) の Elastic Beanstalk オペレーションおよびサンプル AWS アカウント ID 123456789012 に必要な S3 バケットへのアクセスを提供する方法を示しています。

ポリシーの例

> **Important**
>
> このポリシーの特定のリージョンの代わりにワイルドカード文字 (*) を使用しないことをお勧めします。例えば、`arn:aws:s3:::cloudformation-waitcondition-us-east-2/*` を使用して、`arn:aws:s3:::cloudformation-waitcondition-*/*` は使用しないでください。ワイルドカードを使用すると、アクセスを付与する S3 バケットへのアクセス権が付与される場合があります。複数のリージョンでポリシーを使用する場合は、各リージョンの最初の `Statement` ブロックを繰り返すことをお勧めします。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsToCustomerItems",
            "Effect": "Allow",
            "Principal": {"AWS": "*"},
            "Action": ["GetObject",
                       "GetObjectAcl",
                       "PutObject",
                       "PutObjectAcl",
                       "ListBucket"
                       ],
            "Resource": [
                "arn:aws:s3:::elasticbeanstalk-us-east-2-123456789012/*"
            ]
         }
```

```
    ]
}
```

Docker レジストリ認証をサポートするためにカスタマーアカウントが所有する S3 バケット

このバケットは、Docker プラットフォームに基づく環境にのみ適用されます。バケットは、お客様がプロビジョニングした S3 バケットに存在するプライベート Docker レジストリへの認証に使用されるファイルを保存します。詳細については、このガイドの Docker プラットフォームの章の「Dockerrun.aws.json v3 ファイルの使用」を参照してください。

S3 バケット ARN

ARN はカスタマーアカウントによって異なります。

S3 バケット ARN は `arn:aws:s3:::bucket-name` の形式になります。

オペレーション

GetObject

VPC エンドポイントポリシーの例

次の例は、amzn-s3-demo-bucket1 という名前の S3 バケットへのアクセス権を付与する方法を示しています。

ポリシーの例

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsToDockerRegistryAuth",
            "Effect": "Allow",
            "Principal": {"AWS": "*"},
            "Action": ["GetObject"],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1"
            ]
         }
    ]
```

```
}
```

VPC エンドポイントポリシーの更新

VPC エンドポイントには 1 つのポリシーしかアタッチされないため、すべてのアクセス許可を 1 つのポリシーに結合する必要があります。次の例では、前述の例をすべて 1 つにまとめています。

VPC エンドポイントポリシーの作成と使用の詳細については、「AWS PrivateLink ガイド」の「エンドポイントポリシーを使用して VPC エンドポイントへのアクセスを制御する」を参照してください。

前の例と同様に、米国東部 (オハイオ) リージョン (us-east-2) で Elastic Beanstalk の運用に必要な S3 バケットへのアクセス権を付与する方法を示しています。また、サンプル AWS アカウント ID 123456789012 のバケットとサンプルバケット名 amzn-s3-demo-bucket1 のバケットも含まれます。

> **Important**
>
> このポリシーの特定のリージョンの代わりにワイルドカード文字 (*) を使用しないことをお勧めします。例えば、`arn:aws:s3:::cloudformation-waitcondition-us-east-2/*` を使用して、`arn:aws:s3:::cloudformation-waitcondition-*/*` は使用しないでください。ワイルドカードを使用すると、アクセスを付与する S3 バケットへのアクセス権が付与される場合があります。複数のリージョンでポリシーを使用する場合は、各リージョンの最初の `Statement` ブロックを繰り返すことをお勧めします。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsToAWSResources",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "s3:GetObject"
```

```
            ],
            "Resource": [
                "arn:aws:s3:::elasticbeanstalk-platform-assets-us-east-2/*",
                "arn:aws:s3:::elasticbeanstalk-env-resources-us-east-2/*",
                "arn:aws:s3:::elasticbeanstalk-env-resources-us-east-2/*",
                "arn:aws:s3:::elasticbeanstalk-samples-us-east-2/*"
            ]
        },
        {
            "Sid": "AllowRequestsToCloudFormation",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::cloudformation-waitcondition-us-east-2/*",
                "arn:aws:s3:::cloudformation-custom-resource-response-us-east-2/
*"
            ]
        },
        {
            "Sid": "AllowRequestsToCustomerItems",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "GetObject",
                "GetObjectAcl",
                "PutObject",
                "PutObjectAcl",
                "ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::elasticbeanstalk-us-east-2-123456789012/*"
            ]
        },
        {
            "Sid": "AllowRequestsToDockerRegistryAuth",
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": "*"
            },
            "Action": [
                "GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1",
                ""
            ]
        }
    ]
}
```

AWS Elastic Beanstalk セキュリティ

Elastic Beanstalk が担当するセキュリティタスクと、セキュリティとコンプライアンスの目標を達成するために Elastic Beanstalk を使用する場合に考慮する必要があるセキュリティ設定の詳細については、この章で確認してください。

でのクラウドセキュリティが最優先事項 AWS です。 AWS のお客様は、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 とお客様の間で共有される責任です。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して説明しています。

クラウドのセキュリティ — AWS クラウドで提供されるすべてのサービスを実行するインフラストラクチャ AWS を保護し、安全に利用できるサービスを提供します。当社のセキュリティ責任は で最優先事項であり AWS、当社のセキュリティの有効性はAWS 、コンプライアンスプログラムの一環として、サードパーティーの監査者によって定期的にテストおよび検証されています。Elastic Beanstalk に関連する情報については、AWSAWS 保証プログラムの対象となる のサービスを確認してください。

クラウド内のセキュリティ – お客様の責任は、使用している AWS サービス、およびお客様のデータの機密性、組織の要件、適用される法律や規制などのその他の要因によって決まります。このドキュメントは、Elastic Beanstalk を使用する際に、責任共有モデルを適用する方法を理解するのに役立ちます。

トピック

- Elastic Beanstalkのデータ保護
- Elastic Beanstalk の Identity and Access Management
- Elastic Beanstalk でのログ記録とモニタリング
- Elastic Beanstalk のコンプライアンス検証
- Elastic Beanstalk の耐障害性
- Elastic Beanstalk のインフラストラクチャセキュリティ
- Elastic Beanstalk での設定と脆弱性の分析
- Elastic Beanstalk のセキュリティベストプラクティス

Elastic Beanstalk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 Elastic Beanstalk。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 Identity Center 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 が必須で、TLS 1.3 をお勧めします。
- 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 検証済みの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、AWS CLIまたは SDK を使用して Elastic Beanstalk 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。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

トピック

- 暗号化を使用したデータの保護
- インターネット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

Elastic Beanstalk データを保護するために、さまざまな形式のデータ暗号化を使用できます。データ保護とは、転送中 (Elastic Beanstalk との間で送受信されるデータ) および保管中 (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。

転送中の暗号化

転送中のデータ保護は、Secure Sockets Layer (SSL) を使用して接続を暗号化する方法と、クライアント側の暗号化 (オブジェクトが送信される前に暗号化される) を使用する 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。接続をセキュリティで保護するには、アプリケーション、開発者、管理者、エンドユーザーがオブジェクトを送受信するたびに SSL を使用して暗号化します。アプリケーションとの間で送受信されるウェブトラフィックの暗号化の詳細については、「the section called “HTTPS”」を参照してください。

クライアント側の暗号化は、アップロードするアプリケーションバージョンおよびソースバンドルでソースコードを保護するための有効な方法ではありません。Elastic Beanstalk はこれらのオブジェクトにアクセスする必要があるため、暗号化できません。したがって、開発環境またはデプロイ環境と Elastic Beanstalk の間の接続をセキュリティで保護してください。

保管時の暗号化

保管時のアプリケーションのデータを保護するには、アプリケーションが使用するストレージサービスのデータ保護を参照します。例えば、Amazon RDS ユーザーガイドの「Amazon RDS でのデータ保護」、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 におけるデータ保護」、またはAmazon Elastic File System ユーザーガイドの「Encrypting Data and Metadata in EFS」を参照してください。

Elastic Beanstalk は、環境を作成する AWS リージョンごとに作成する暗号化された Amazon Simple Storage Service (Amazon S3) バケットにさまざまなオブジェクトを保存します。Elastic Beanstalk は Amazon S3 が提供するデフォルトの暗号化を保持するため、暗号化された Amazon S3 バケットを作成します。詳細については、「the section called “Amazon S3”」を参照してください。保存

されたオブジェクトの一部を指定し、Elastic Beanstalk に送信します (アプリケーションバージョンやソースバンドルなど)。Elastic Beanstalk は、ログファイルなどの他のオブジェクトを生成します。Elastic Beanstalk が保存するデータに加えて、アプリケーションはオペレーションの一部としてデータを転送したり、保存したりすることができます。

環境のインスタンスにアタッチされた Amazon Elastic Block Store (Amazon EBS) ボリュームに保存されているデータを保護するには、アカウントとリージョンで Amazon EBS 暗号化をデフォルトで有効にします AWS 。有効にすると、すべての新しい Amazon EBS ボリュームとそのスナップショットは、 AWS Key Management Service キーを使用して自動的に暗号化されます。詳細については、「Amazon EBS ユーザーガイド」の「デフォルトでの暗号化」を参照してください。

データ保護の詳細については、AWS セキュリティブログ のブログ投稿「AWS の責任共有モデルと GDPR」を参照してください。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

インターネットトラフィックのプライバシー

Amazon Virtual Private Cloud (Amazon VPC) を使用して、Elastic Beanstalk アプリケーションのリソース間に境界を作成し、それらの間のトラフィック、オンプレミスネットワーク、インターネットを制御できます。詳細については、「the section called “Amazon VPC”」を参照してください。

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「セキュリティ」を参照してください。

データ保護の詳細については、AWS セキュリティブログ のブログ投稿「AWS の責任共有モデルと GDPR」を参照してください。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

Elastic Beanstalk 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するのに役立つ AWS サービスです。IAM 管理者は、誰を認証 (サインイン) し、誰に AWS Elastic Beanstalk リソースの使用を許可する (権限を持たせる) かを制御します。IAM は、追加料金なしで使用できる AWS サービスです。

IAM の操作の詳細については、「AWS Identity and Access Managementで Elastic Beanstalk を使用する」を参照してください。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

AWS の マネージドポリシー AWS Elastic Beanstalk

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。

Elastic Beanstalk の AWS マネージドポリシーの更新

2021 年 3 月 1 日以降の Elastic Beanstalk の AWS マネージドポリシーの更新に関する詳細を表示します。

特定の管理ポリシーの JSON ソースを確認するには、「AWS 管理ポリシーリファレンスガイド」を参照してください。

変更	説明	日付
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy - 更新済みの既存のポリシー	このポリシーは、 が有効になっている場合に Elastic Beanstalk 起動テンプレートへのタグの伝播がマネージド更	2025 年 2 月 27 日

変更	説明	日付
	新を実行できるように更新されました。 詳細については、「マネージドサービスロールのポリシー」を参照してください。	
AdministratorAccess-AWSElasticBeanstalk – 既存のポリシーを更新しました	このポリシーは、StringLike 演算子を ArnLike 演算子に置き換えて、条件ブロック の ARN タイプのキーを評価するように更新されましたiam:PolicyArn 。これにより、より安全な適用が可能になります。 詳細については、「Elastic Beanstalk ユーザーポリシーの管理」を参照してください。	2024 年 12 月 11 日

変更	説明	日付
次のポリシーが更新されました。 • AWSElasticBeanstalkInternalMaintenanceRolePolicy • AWSElasticBeanstalkMaintenance • AWSElasticBeanstalkManagedUpdatesInternalServiceRolePolicy • AWSElasticBeanstalkManagedUpdatesServiceRolePolicy • AWSElasticBeanstalkRoleCore	これらのポリシーは、 AWS CloudFormation スタックまたは変更セットを作成または更新するときに Elastic Beanstalk がタグを追加または削除できるように更新されました。 `AWSElasticBeanstalkManagedUpdatesServiceRolePolicy` の詳細については、Elastic Beanstalk のサービスにリンクされたロールのアクセス許可 を参照してください。 `AWSElasticBeanstalkRoleCore` の詳細については、他のサービスとの統合に関するポリシー を参照してください。	2024 年 4 月 30 日

変更	説明	日付
AWSElasticBeanstalkService - 更新された既存のポリシー	このポリシーは、Elastic Load Balancing、Auto Scaling グループ (ASG)、Amazon ECS 用にリソースが作成される際に Elastic Beanstalk にタグ付けを許可するため、更新されました。 **Note** このポリシーは `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` によって以前に置き換えられました。このポリシーは、新しい IAM ユーザー、グループ、ロールにアタッチできなくなりましたが、まだ既存のものにアタッチできます。 詳細については、「マネージドサービスロールのポリシー」を参照してください。	2023 年 5 月 10 日

変更	説明	日付
AWSElasticBeanstalkMulticontainerDocker – 既存のポリシーを更新しました	このポリシーは、Amazon ECS 用にリソースが作成される際に Elastic Beanstalk にタグ付けを許可するため、更新されました。 詳細については、「Elastic Beanstalk インスタンスプロファイルの管理」を参照してください。	2023 年 3 月 23 日
AWSElasticBeanstalkRoleECS – 既存のポリシーを更新しました	このポリシーは、Amazon ECS 用にリソースが作成される際に Elastic Beanstalk にタグ付けを許可するため、更新されました。 詳細については、「他のサービスとの統合に関するポリシー」を参照してください。	2023 年 3 月 23 日
AdministratorAccess-AWSElasticBeanstalk – 既存のポリシーを更新しました	このポリシーは、Amazon ECS 用にリソースが作成される際に Elastic Beanstalk にタグ付けを許可するため、更新されました。 詳細については、「Elastic Beanstalk ユーザーポリシーの管理」を参照してください。	2023 年 3 月 23 日

変更	説明	日付
AWSElasticBeanstalkManagedUpdatesServiceRolePolicy – 更新された既存のポリシー	このポリシーは、Elastic Beanstalk が Amazon ECS リソースを作成する際にタグを追加することを許可するように更新されました。 詳細については、「Elastic Beanstalk のサービスにリンクされたロールのアクセス許可」を参照してください。	2023 年 3 月 23 日
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy – 更新された既存のポリシー	このポリシーは、Elastic Beanstalk が Amazon ECS リソースを作成する際にタグを追加することを許可するように更新されました。 詳細については、「マネージドサービスロールのポリシー」を参照してください。	2023 年 3 月 23 日
AWSElasticBeanstalkManagedUpdatesServiceRolePolicy – 更新された既存のポリシー	このポリシーは、Elastic Beanstalk が Auto Scaling グループの作成時にタグを追加することを許可するように更新されました。 詳細については、「マネージド更新サービスにリンクされたロール」を参照してください。	2023 年 1 月 27 日

変更	説明	日付
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy – 更新された既存のポリシー	このポリシーは、Elastic Beanstalk が Auto Scaling グループ (ASG) の作成時にタグを追加できるように更新されました。 詳細については、「マネージドサービスロールのポリシー」を参照してください。	2023 年 1 月 23 日
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy - 更新済みの既存のポリシー	このポリシーは、Elastic Beanstalk で Elastic Load Balancing (ELB) の作成時にタグを追加できるように更新されました。 詳細については、「マネージドサービスロールのポリシー」を参照してください。	2022 年 12 月 21 日

変更	説明	日付
AWSElasticBeanstalkManagedUpdatesServiceRolePolicy – 更新された既存のポリシー	マネージド更新中に Elastic Beanstalk が次の処理を行うための許可がこのポリシーに追加されました。 • 起動テンプレートとテンプレートバージョンの作成と削除。 • 起動テンプレートを使用した Amazon EC2 インスタンスの起動。 • (Amazon RDS が存在する場合) 使用可能な DB エンジンのリスト、およびプロビジョニングされた RDS インスタンスに関する情報の取得 詳細については、「マネージド更新サービスにリンクされたロール」を参照してください。	2022 年 8 月 23 日

変更	説明	日付
AWSElasticBeanstalkReadOnlyAccess – 非推奨 GovCloud (米国) AWS リージョン	このポリシーは `AWSElasticBeanstalkReadOnly` に置き換えられました。 このポリシーは GovCloud (米国) で段階的に廃止されます AWS リージョン。 廃止されると、2021 年 6 月 18 日以降、このポリシーで新しい IAM ユーザー、グループ、ロールにアタッチできなくなります。 詳細については、「ユーザーポリシー」を参照してください。	2021 年 6 月 17 日
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy - 更新済みの既存のポリシー	このポリシーは、Elastic Beanstalk で EC2 アベイラビリティーゾーンの属性を読み取れるように更新されました。これにより Elastic Beanstalk では、アベイラビリティーゾーン全体におけるインスタンスタイプの選択をより効果的に検証できます。 詳細については、「マネージドサービスロールのポリシー」を参照してください。	2021年6月16日

変更	説明	日付
AWSElasticBeanstalkFullAccess – 非推奨 GovCloud (米国) AWS リージョン	このポリシーは `AdministratorAccess-AWSElasticBeanstalk` に置き換えられました。 このポリシーは GovCloud (米国) で段階的に廃止されます AWS リージョン。 廃止されると、2021 年 6 月 11 日以降、このポリシーで新しい IAM ユーザー、グループ、ロールにアタッチできなくなります。 詳細については、「ユーザーポリシー」を参照してください。	2021 年 6 月 10 日

変更	説明	日付
以下の マネージドポリシーは、すべての中国 AWS リージョンで廃止されました。 • AWSElasticBeanstalkFullAccess • AWSElasticBeanstalkReadOnlyAccess	`AWSElasticBeanstalkFullAccess` ポリシーは `AdministratorAccess-AWSElasticBeanstalk` に置き換えられました。 `AWSElasticBeanstalkReadOnlyAccess` ポリシーは `AWSElasticBeanstalkReadOnly` に置き換えられました。 これらのポリシーはすべての中国で段階的に廃止されました AWS リージョン。 2021 年 6 月 4 日以降、これらのポリシーで新しい IAM ユーザー、グループ、またはロールにアタッチできなくなります。 詳細については、「ユーザーポリシー」を参照してください。	2021 年 6 月 3 日

変更	説明	日付
AWSElasticBeanstalkService – 非推奨	このポリシーは `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` によって置き換えられました。 このポリシーは廃止され、新しい IAM ユーザー、グループ、またはロールにアタッチできなくなりました。 詳細については、「マネージドサービスロールのポリシー」を参照してください。	2021 年 6 月 - 2022 年 1 月

変更	説明	日付
以下の マネージドポリシーは、中国および GovCloud (米国) を除くすべての AWS リージョンで廃止されました。 • AWSElasticBeanstalkFullAccess • AWSElasticBeanstalkReadOnlyAccess	`AWSElasticBeanstalkFullAccess` ポリシーは `AdministratorAccess-AWSElasticBeanstalk` に置き換えられました。 `AWSElasticBeanstalkReadOnlyAccess` ポリシーは `AWSElasticBeanstalkReadOnly` に置き換えられました。 これらのポリシーは、中国と GovCloud (米国) を除くすべての AWS リージョンで段階的に廃止されました。 2021 年 4 月 17 日以降、これらのポリシーで新しい IAM ユーザー、グループ、またはロールにアタッチできなくなります。 詳細については、「ユーザーポリシー」を参照してください。	2021 年 4 月 16 日

変更	説明	日付
次の管理ポリシーが更新されました。 • AdministratorAccess-AWSElasticBeanstalk • AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy	これらのポリシーはどちらも、中国で PassRole アクセス許可をサポートするようになりました AWS リージョン。 `AdministratorAccess-AWSElasticBeanstalk` の詳細については、「ユーザーポリシー」を参照してください。 `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` の詳細については、「マネージドサービスロールのポリシー」を参照してください。	2021 年 3 月 9 日
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy – 新しいポリシー	Elastic Beanstalk は `AWSElasticBeanstalkService` 管理ポリシーを、追加された新しいポリシーと置き換えました。 この新しい管理ポリシーでは、より制限の厳しいアクセス許可セットが適用されるので、リソースのセキュリティが向上されます。 詳細については、「マネージドサービスロールのポリシー」を参照してください。	2021 年 3 月 3 日

変更	説明	日付
Elastic Beanstalk に変更の追跡が追加	Elastic Beanstalk 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

Elastic Beanstalk でのログ記録とモニタリング

AWS には、Elastic Beanstalk リソー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ツールがいくつか用意されています。モニタリングは、 および AWS Elastic Beanstalk AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 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必要があります。

モニタリングの詳細については、「Elastic Beanstalk での環境のモニタリング」をご参照ください。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

拡張ヘルスレポート

拡張ヘルスレポートは、環境で有効にすることができる機能の 1 つであり、これにより、Elastic Beanstalk は環境内のリソースに関する追加の情報を収集できます。Elastic Beanstalk は、情報を分析して環境全体の状態をより的確に示し、アプリケーションの使用を妨げる可能性のある問題を特定します。詳細については、「Elastic Beanstalk でのヘルスレポートとモニタリングの強化」を参照してください。

Amazon EC2 インスタンス名

Elastic Beanstalk 環境の Amazon EC2 インスタンスでは、アプリケーションまたは設定ファイルに関する問題を解決する際に確認するためのログが生成されます。ウェブサーバー、アプリケーションサーバー、Elastic Beanstalk プラットフォームスクリプト、 によって作成されたログ AWS CloudFormation は、個々のインスタンスにローカルに保存されます。それらは、環境管理コンソールまたは EB CLI を使用して簡単に検索できます。また、Amazon CloudWatch Logs にリアルタイムでログがストリーミングされるように環境を設定することもできます。詳細については、「Elastic Beanstalk 環境の Amazon EC2 インスタンスからのログの表示」を参照してください。

環境の通知

Elastic Beanstalk 環境は、Amazon Simple Notification Service (Amazon SNS) を使用して、アプリケーションに影響を与える重要なイベントを通知するように設定できます。環境の作成中または作成後に E メールアドレスを指定して、エラーが発生した AWS とき、または環境のヘルスが変化したときに から E メールを受信します。詳細については、「Amazon SNS を使用した Elastic Beanstalk 環境の通知」を参照してください。

Amazon CloudWatch アラーム

Amazon CloudWatch アラームを使用して、指定した期間にわたって 1 つのメトリクスを確認します。メトリクスが特定のしきい値を超えると、Amazon SNS トピックまたは AWS Auto Scaling ポリシーに通知が送信されます。CloudWatch アラームは、特定の状態にあるという理由ではアクションを呼び出しません。代わりに、状態が変わり、指定した期間数にわたってこの状態が維持されたときに、アラームによってアクションが呼び出されます。詳細については、「「Amazon CloudWatch で Elastic Beanstalk を使用する」」を参照してください。

AWS CloudTrail ログ

CloudTrail は、Elastic Beanstalk のユーザー、ロール、または AWS サービスによって実行されたアクションの記録を提供します。CloudTrail で収集された情報を使用して、Elastic Beanstalk 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。詳細については、「を使用した Elastic Beanstalk API コールのログ記録 AWS CloudTrail」を参照してください。

AWS X-Ray デバッグ

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し、それを使用して、アプリケーションの問題と最適化の機会を特定するために使用できるサービスマップを構築する AWS サービスです。 AWS Elastic Beanstalk コンソールまたは設定ファイルを使用して、環境内のインスタンスで X-Ray デーモンを実行できます。詳細については、「AWS X-Ray デバッグの設定」を参照してください。

Elastic Beanstalk のコンプライアンス検証

のセキュリティとコンプライアンス AWS Elastic Beanstalk は、複数の AWS コンプライアンスプログラムの一環として、サードパーティーの監査者によって評価されます。これには、SOC、PCI、FedRAMP、HIPAA などが含まれます。 は、コンプライアンスプログラムによる対

象範囲内の AWS のサービスで、特定のコンプライアンスプログラムの対象となる のサービスの頻繁な更新リスト AWS を提供します。 AWS

サードパーティーの監査レポートは、 を使用してダウンロードできます AWS Artifact。詳細については、「 でレポートをダウンロードする AWS Artifact」を参照してください。

AWS コンプライアンスプログラムの詳細については、AWS 「 コンプライアンスプログラム」を参照してください。

Elastic Beanstalk を使用する際のお客様のコンプライアンス責任は、組織のデータの機密性や組織のコンプライアンス目的、適用可能な法律、規制によって決定されます。Elastic Beanstalk の使用が HIPAA、PCI、FedRAMP などの標準に準拠していることを前提としている場合、 は以下を支援するリソース AWS を提供します。

- セキュリティとコンプライアンスのクイックスタートガイド – アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境をデプロイする手順を提供するデプロイガイド AWS。
- アマゾン ウェブ サービスでの HIPAA セキュリティとコンプライアンスのためのアーキテクチャ – 企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法を説明するホワイトペーパー。
- AWS コンプライアンスリソース – お客様の業界や地域に適用される可能性のあるコンプライアンスワークブックとガイドのコレクション。
- AWS Config – 自社プラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価するサービス。
- AWS Security Hub – セキュリティ業界標準とベストプラクティスへの準拠を確認するのに役立つ AWS 、 内のセキュリティ状態の包括的なビュー。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

Elastic Beanstalk の耐障害性

AWS Elastic Beanstalk 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。Elastic Beanstalk 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のメリットが得られます。

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。

AWS リージョンは、低レイテンシー、高スループット、および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 リージョンと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。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

Elastic Beanstalk のインフラストラクチャセキュリティ

マネージドサービスである AWS Elastic Beanstalk は、セキュリティ、アイデンティティ、コンプライアンスのベストプラクティスのウェブサイトに記載されている AWS グローバルネットワークセキュリティの手順で保護されています。

が AWS 公開した API コールを使用して、ネットワーク経由で Elastic Beanstalk にアクセスします。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、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。最新のプラットフォームは、ほとんどの場合これらのモードをサポートしています。

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。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

Elastic Beanstalk で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。 AWS Elastic Beanstalk は、マネージド更新機能を提供することで、責任共有モデルのユーザー側を実行するのに役立ちます。この機能では、Elastic Beanstalk でサポートされているプラットフォームのバージョンに対してパッチとマイナーな更新を自動的に適用します。

詳細については、「Elastic Beanstalk プラットフォームメンテナンスの責任共有モデル」を参照してください。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

Elastic Beanstalk のセキュリティベストプラクティス

AWS Elastic Beanstalk 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。

Elastic Beanstalk のその他のセキュリティトピックについては、「AWS Elastic Beanstalk セキュリティ」を参照してください。

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

Elastic Beanstalk は、インスタンスプロファイル、サービスロール、IAM ユーザーの 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では、環境とアプリケーションの正しいオペレーションに必要なすべてのアクセス権限を指定します。

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。カスタマイズにより、環境のインスタンス、Elastic Beanstalk サービス、およびユーザー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。

機密性の高いアプリケーションデータを保護する

アプリケーションが認証情報、API キー、設定データなどの機密情報にアクセスする必要がある場合は、以下の手順に従ってセキュリティを維持します。

- アプリケーションコード内のそれぞれの SDKs AWS Secrets Manager または APIs を使用して、または AWS Systems Manager Parameter Store から直接機密データを取得します。これにより、最も安全で柔軟な方法で機密情報にアクセスできます。

- AWS Secrets Manager または Parameter Store AWS Systems Manager から環境変数として機密データを渡す場合 (「」を参照環境変数へのシークレットの取得)、EC2 キーペアへのアクセスを慎重に制限し、インスタンスの最小特権のアクセス許可を持つ適切な IAM ロールを設定します。
- アプリケーションコードで機密データを出力、ログ記録、または公開しないでください。これらの値は、権限のないユーザーに表示される可能性のあるログファイルやエラーメッセージになる可能性があります。

プラットフォームを定期的に更新する

Elastic Beanstalk は、新しいプラットフォームバージョンを定期的にリリースして、すべてのプラットフォームを更新します。新しいプラットフォームバージョンでは、オペレーティングシステム、ランタイム、アプリケーションサーバー、ウェブサーバーの更新、Elastic Beanstalk コンポーネントの更新が提供されます。これらのプラットフォーム更新の多くには、重要なセキュリティ修正が含まれています。Elastic Beanstalk 環境が、サポートされているプラットフォームバージョン (通常はプラットフォームの最新バージョン) で実行されていることを確認してください。詳細については、「Elastic Beanstalk 環境のプラットフォームバージョンの更新」を参照してください。

環境のプラットフォームを最新の状態に保つ最も簡単な方法は、マネージドプラットフォーム更新を使用するように環境を設定することです。

環境インスタンスに IMDSv2 を適用する

Elastic Beanstalk 環境の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスは、インスタンス上のコンポーネントである Instance Metadata Service (IMDS) を使用して、インスタンスメタデータに安全にアクセスします。IMDS は、IMDSv1 と IMDSv2 という 2 つのデータアクセス手法をサポートしています。IMDSv2 はセッション指向のリクエストを使用し、IMDS へのアクセス試行に利用される可能性があるいくつかのタイプの脆弱性を軽減します。IMDSv2 の利点の詳細については、EC2 Instance Metadata Service に多層防御を追加する拡張機能のページを参照してください。

IMDSv2 の方が安全性に優れているため、インスタンスには IMDSv2 を適用するようお勧めします。IMDSv2 を適用するには、アプリケーションのすべてのコンポーネントが IMDSv2 をサポートしていることを確認してから、IMDSv1 を無効にします。詳細については、「the section called “IMDS”」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、Elastic Beanstalk 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。 AWS には、 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が用意されています。

以下は、モニタリングする項目のいくつかの例です。

- Elastic Beanstalk の Amazon CloudWatch メトリクス – 主要な Elastic Beanstalk 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、「「Amazon CloudWatch で Elastic Beanstalk を使用する」」を参照してください。
- AWS CloudTrail エントリ – `UpdateEnvironment`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`TerminateEnvironment`。詳細については、「を使用した Elastic Beanstalk API コールのログ記録 AWS CloudTrail」を参照してください。

有効 AWS Config

AWS Config は、アカウント内の AWS リソースの設定の詳細ビューを提供します。リソース間の関係、設定変更の履歴、関係と設定の時間的な変化を確認できます。

を使用して AWS Config 、データコンプライアンスのリソース設定を評価するルールを定義できます。 AWS Config ルールは、Elastic Beanstalk リソースの理想的な設定を表します。リソースがルールに違反し、非準拠としてフラグが付けられている場合、Amazon Simple Notification Service (Amazon SNS) トピックを使用してアラートを送信 AWS Config できます。詳細については、を使用した Elastic Beanstalk リソースの検索と追跡 AWS Config を参照してください

Elastic Beanstalk サービスロール、インスタンスプロファイル、ユーザーポリシー

ロールは、アクセス許可を適用するために AWS Identity and Access Management (IAM) で作成するエンティティです。Elastic Beanstalk 環境が正しく機能するために必要なロールがあります。また、ユーザーまたはグループに割り当てることができる独自のカスタムポリシーとロールを作成することもできます。

Elastic Beanstalk 環境に必要なロール

環境を作成すると、 によって次の AWS Identity and Access Management (IAM) ロールを指定するように AWS Elastic Beanstalk 求められます。

- サービスロール: Elastic Beanstalk は、ユーザーに代わって他の を使用するサービスロールを引き受け AWS のサービス ます。
- インスタンスプロファイル: Elastic Beanstalk は、環境内の Amazon EC2 インスタンスにインスタンスプロファイルを適用します。このアクションにより、Amazon Simple Storage Service (Amazon S3) からの情報の取得や S3 へのログのアップロードなど、必要なタスクを実行できます。

サービスロールと EC2 インスタンスプロファイルロールを作成する

AWS アカウントに EC2 インスタンスプロファイルまたはサービスロールがない場合は、IAM サービスを使用してそれぞれ 1 つを作成する必要があります。その後、作成した新しい環境に EC2 インスタンスプロファイルとサービスロールを割り当てることができます。環境の作成ウィザードは IAM サービスに誘導するため、必要なアクセス許可でこれらのロールを作成できます。

Elastic Beanstalk 環境を管理するためのオプションのポリシーとロール

オプションでユーザーポリシーを作成し、アカウント内の IAM ユーザーとグループに適用できます。ユーザーポリシーを適用すると、ユーザーは Elastic Beanstalk アプリケーションと環境を作成および管理できます。また、Elastic Beanstalk 管理ポリシーを割り当てて、ユーザーまたはグループにフルアクセスと読み取り専用アクセスを許可することもできます。これらのポリシーの詳細については、「the section called “ユーザーポリシー”」を参照してください。

高度なシナリオ用に独自のインスタンスプロファイルとユーザーポリシーを作成できます。インスタンスからデフォルトのポリシーに含まれていないサービスにアクセスする必要がある場合は、新しいポリシーを作成するか、デフォルトのポリシーにポリシーを追加することができます。管理ポリシーの許容範囲がニーズに対して広すぎる場合は、範囲を絞り込んだユーザーポリシーを作成することもできます。アクセス AWS 許可の詳細については、IAM ユーザーガイドを参照してください。

Elastic Beanstalk サービスロール

サービスロールは、他のサービスを代理で呼び出すときに Elastic Beanstalk が引き受ける IAM ロールです。例えば、Elastic Beanstalk は、Amazon Elastic Compute Cloud (Amazon EC2)、Elastic Load Balancing、Amazon EC2 Auto Scaling の各 API を呼び出すときに、サービスロールを使用して情報を収集します。Elastic Beanstalk が使用するサービスロールは、Elastic Beanstalk 環境を作成するときに指定したサービスロールになります。

サービスロールにアタッチする管理ポリシーには 2 つの種類があります。これらのポリシーは、Elastic Beanstalk が環境を作成および管理するために必要な AWS リソースにアクセスできるようにするアクセス許可を提供します。1 つは、拡張ヘルスモニタリングと、ワーカー層の Amazon SQS サポートに必要なアクセス権限を付与する管理ポリシーであり、もう 1 つは、マネージドプラットフォームの更新に必要な追加のアクセス権限を付与する管理ポリシーです。

AWSElasticBeanstalkEnhancedHealth

このポリシーでは、インスタンスおよび環境のヘルスをモニタリングするアクセス許可を Elastic Beanstalk に付与します。このポリシーには、Elastic Beanstalk にワーカー環境のキューアクティビティのモニタリングを許可する Amazon SQS アクションも含まれています。この管理ポリシーの内容を確認するには、AWS 「 管理ポリシーリファレンスガイド」のAWSElasticBeanstalkEnhancedHealth」ページを参照してください。

AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy

このポリシーでは、お客様に代わって環境を更新してマネージドプラットフォームを更新するアクセス許可を Elastic Beanstalk に付与します。この管理ポリシーの内容を確認するには、AWS 「 管理ポリシーリファレンスガイド」のAWSElasticBeanstalkManagedUpdatesCustomerRolePolicy」ページを参照してください。

サービスレベルでのアクセス許可のグループ化

このポリシーは、提供された一連の許可に基づくステートメントごとにグループ化されます。

- *ElasticBeanstalkPermissions* – このアクセス許可グループは、Elastic Beanstalk サービスアクション (Elastic Beanstalk API) を呼び出す際に使用します。
- *AllowPassRoleToElasticBeanstalkAndDownstreamServices* – このアクセス許可グループにより、Elastic Beanstalk や AWS CloudFormationなどのダウンストリームサービスに対し、任意のロールを渡すことが可能になります。
- *ReadOnlyPermissions* – このアクセス許可グループは、実行中の環境に関する情報を収集するため使用します。
- **OperationPermissions* – この命名パターンを持つグループは、プラットフォームの更新を実行するために必要なオペレーションを呼び出すためのものです。
- **BroadOperationPermissions* – この命名パターンを持つグループは、プラットフォームの更新を実行するために必要なオペレーションを呼び出すためのものです。またこれには、レガシー環境をサポートするための広範なアクセス許可も含まれています。
- `*TagResource`– この命名パターンのグループは、tag-on-create API を使用して Elastic Beanstalk 環境で作成されているリソースにタグをアタッチする呼び出し用です。

次のいずれかの方法で Elastic Beanstalk 環境を作成できます。各セクションでは、このアプローチがサービスロールを処理する方法について説明します。

Elastic Beanstalk コンソール

Elastic Beanstalk コンソールを使って環境を作成する場合、Elastic Beanstalk で、`aws-elasticbeanstalk-service-role` という名前のサービスロールを作成するよう指示されます。Elastic Beanstalk を介して作成されたこのロールには、Elastic Beanstalk がそのサービスロールを担うことを許可する信頼ポリシーが含まれます。このトピックで前述した 2 つの管理ポリシーもこのロールにアタッチされています。

Elastic Beanstalk コマンドラインインターフェイス (EB CLI)

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) の the section called “eb create” コマンドを使用して環境を作成できます。`--service-role` オプションを使用してサービスロールを指定しない場合は、Elastic Beanstalk が同じデフォルトサービスロール `aws-elasticbeanstalk-service-role` を作成します。デフォルトのサービスロールが既に存在する場合、Elastic Beanstalk はそのサービスロールを新しい環境で使用します。Elastic Beanstalk を介して作成されたこのロールには、Elastic Beanstalk がそのサービスロールを担うことを許可する信頼ポリシーが含まれます。このトピックで前述した 2 つの管理ポリシーもこのロールにアタッチされています。

Elastic Beanstalk API

Elastic Beanstalk API の `CreateEnvironment` アクションを使用して環境を作成することができます。サービスロールを指定しない場合、Elastic Beanstalk がモニタリングサービスリンクロールを作成します。これは、サービスが AWS のサービス ユーザーに代わって他の を呼び出すために必要なすべてのアクセス許可を含めるために Elastic Beanstalk によって事前定義された一意のタイプのサービスロールです。このサービスリンクロールはお客様のアカウントに関連付けられています。Elastic Beanstalk によって一度作成されたロールは、追加の環境を作成するときにも使用されます。IAM を使用して、アカウント用のモニタリングサービスリンクロールを事前に作成することもできます。アカウントにモニタリングサービスリンクロールがある場合は、Elastic Beanstalk コンソール、Elastic Beanstalk API、または EB CLI のいずれかを使用して、そのロールで環境を作成することができます。サービスリンクロールを Elastic Beanstalk 環境で使用する方法については、「Elastic Beanstalk でのサービスにリンクされたロールの使用」を参照してください。

サービスロールの詳細については、「Elastic Beanstalk サービスロールの管理」を参照してください。

Elastic Beanstalk インスタンスプロファイル

インスタンスプロファイルは、Elastic Beanstalk 環境で起動される Amazon EC2 インスタンスに適用される IAM ロールです。Elastic Beanstalk 環境を作成するとき、EC2 インスタンスが次のアクションを実行する際に使用するインスタンスプロファイルを指定します。

- Amazon Simple Storage Service (Amazon S3) からのアプリケーションバージョンの取得
- Amazon S3 へのログの書き込み
- AWS X-Ray 統合環境で、デバッグデータを X-Ray にアップロードします。
- Amazon ECS マネージド Docker 環境における Amazon Elastic Container Service (Amazon ECS) を使用したコンテナデプロイの調整
- ワーカー環境で、Amazon Simple Queue Service (Amazon SQS) キューから読み込む
- ワーカー環境で、Amazon DynamoDB によりリーダーを選択する
- ワーカー環境で Amazon CloudWatch にインスタンスヘルスメトリクスを発行する

マネージドポリシー

Elastic Beanstalk には、環境内の EC2 インスタンスが必要なオペレーションの実行を許可する一連の管理ポリシーが用意されています。基本的なユースケースに必要な管理ポリシーは次のとおりです。

- `AWSElasticBeanstalkWebTier`
- `AWSElasticBeanstalkWorkerTier`
- `AWSElasticBeanstalkMulticontainerDocker`

ウェブアプリケーションが他の追加にアクセスする必要がある場合は AWS のサービス、それらのサービスへのアクセスを許可するステートメントまたは管理ポリシーをインスタンスプロファイルに追加します。詳細については、「デフォルトのインスタンスプロファイルにアクセス許可を追加する」を参照してください。

EC2 インスタンスプロファイルの作成

AWS アカウントに EC2 インスタンスプロファイルがない場合は、IAM サービスを使用して作成する必要があります。その後、作成する新しい環境に EC2 インスタンスプロファイルを割り当てることができます。Elastic Beanstalk コンソールの環境の作成ステップでは、必要なアクセス許可を持つ EC2 インスタンスプロファイルを作成できるように、IAM コンソールにアクセスできます。

Elastic Beanstalk コンソールを経由せずに、IAM コンソールに直接アクセスして EC2 インスタンスプロファイルを作成することもできます。IAM コンソールで Elastic Beanstalk EC2 インスタンスプロファイルを作成する詳細な手順については、「」を参照してくださいインスタンスプロファイルの作成。

Elastic Beanstalk ユーザーポリシー

ルートアカウントの使用や認証情報の共有を回避するには、Elastic Beanstalk を使用する各ユーザーに IAM ユーザーを作成します。セキュリティに関するベストプラクティスとして、これらのユーザーには、彼らが必要とするサービスと機能へのアクセスのみを許可するようにします。

Elastic Beanstalk には、独自の API アクションに対してだけでなく、他のいくつかの AWS のサービスに対してもアクセス許可が必要です。Elastic Beanstalk はユーザーのアクセス許可を使用して、環境内でリソースを起動します。これらのリソースには、EC2 インスタンス、Elastic Load Balancing ロードバランサー、Auto Scaling グループが含まれます。また、Elastic Beanstalk は、Amazon Simple Storage Service (Amazon S3) へのログとテンプレートの保存、Amazon SNS への通知の送信、インスタンスプロファイルの割り当て、および CloudWatch へのメトリクスのパブリッシュにもユーザーアクセス許可を使用します。Elastic Beanstalk には、リソースのデプロイと更新をオーケストレーションするための AWS CloudFormation アクセス許可が必要です。また、必要に応じてデータベースを作成するには Amazon RDS アクセス許可が必要で、ワーカー環境のキューを作成するには Amazon SQS アクセス許可が必要です。

ユーザーポリシーの詳細については、「Elastic Beanstalk ユーザーポリシーの管理」を参照してください。

チュートリアルおよびサンプル

言語とフレームワーク固有のチュートリアルは、 AWS Elastic Beanstalk デベロッパーガイド全体に分散されています。新規および更新されたチュートリアルが発行されたら、この一覧に追加されます。最新のものが最初に表示されます。

これらのチュートリアルは中級ユーザーを対象としており、 AWSへのサインアップなどの基本的な手順の説明は含まれない場合があります。 AWS または Elastic Beanstalk を初めて使用する場合は、「開始方法」のチュートリアルを参照して、最初の Elastic Beanstalk 環境を起動して実行してください。

- Ruby on Rails - Elastic Beanstalk への Rails アプリケーションのデプロイ
- Ruby と Sinatra - Elastic Beanstalk への Sinatra アプリケーションのデプロイ
- PHP と MySQL HA 設定 - 外部 Amazon RDS データベースを使用して高可用性の PHP アプリケーションを Elastic Beanstalk にデプロイする
- PHP と Laravel - Elastic Beanstalk への Laravel アプリケーションのデプロイ
- PHP と CakePHP - Elastic Beanstalk への CakePHP アプリケーションのデプロイ
- PHP と Drupal HA 設定 - 外部 Amazon RDS データベースを使用して高可用性の Drupal ウェブサイトを Elastic Beanstalk にデプロイする
- PHP と WordPress HA 設定 - 外部 Amazon RDS データベースを使用して高可用性の WordPress ウェブサイトを Elastic Beanstalk にデプロイする
- Node.js と DynamoDB HA 設定 - DynamoDB を使用して Node.js アプリケーションを Elastic Beanstalk にデプロイする
- ASP.NET Core - QuickStart: Elastic Beanstalk に ASP.NET アプリケーションをデプロイする
- Python と Flask - Elastic Beanstalk への Flask アプリケーションのデプロイ
- Python と Django - Elastic Beanstalk への Django アプリケーションのデプロイ
- Node.js と Express - Node.js Express アプリケーションの Elastic Beanstalk へのデプロイ
- Docker、PHP、nginx - Elastic Beanstalk コンソールを使用した ECS マネージド Docker 環境の作成

以下のリンクでソースバンドルを提供することなく環境を作成するときは、Elastic Beanstalk によって使用されるサンプルアプリケーションをダウンロードできます。

- Docker - docker.zip

- 複数コンテナ Docker – docker-multicontainer-v2.zip
- 事前設定済み Docker (Glassfish) – docker-glassfish-v1.zip
- Go – go.zip
- Corretto – corretto.zip
- Tomcat – tomcat.zip
- Linux 上の .NET Core – dotnet-core-linux.zip
- .NET Core – dotnet-asp-windows.zip
- Node.js – nodejs.zip
- PHP – php.zip
- Python – python.zip
- Ruby – ruby.zip

追加のウェブフレームワーク、ライブラリ、ツールの使用方法を示す複雑なサンプルアプリケーションは GitHub でオープンソースプロジェクトとして入手できます。

- ロードバランシングされた WordPress (チュートリアル) - WordPress を安全にインストールし、ロードバランシングされた Elastic Beanstalk 環境で実行するための設定ファイル。
- ロードバランシングされた Drupal (チュートリアル) - Drupal を安全にインストールし、ロードバランシングされた Elastic Beanstalk 環境で実行するための設定ファイルと手順。
- Scorekeep - Spring フレームワークと を使用して、ユーザー、セッション、ゲームを作成および管理するためのインターフェイス AWS SDK for Java を提供する RESTful ウェブ API。API は、HTTP を介して API を使用する Angular 1.5 ウェブアプリにバンドルされています。Amazon Cognito、 AWS X-Ray、および Amazon Relational Database Service との統合を示すブランチが含まれます。

  アプリケーションは、Java SE プラットフォームの機能を使って依存関係をダウンロードし、オンインスタンスを構築することで、ソースバンドルのサイズを最小化します。また、アプリケーションには、プロキシを通じてポート 80 で静的にフロントエンドウェブアプリに対応するデフォルト設定を上書きし、`/api` で実行される API に `localhost:5000` 以下のパスをルーティングする nginx 設定ファイルが含まれています。
- Does it Have Snakes? - Elastic Beanstalk で実行される Java EE ウェブアプリケーションでの RDS の使用方法を示す Tomcat アプリケーションです。このプロジェクトでは、サーブレット、JSP、簡易タグのサポート、タグファイル、JDBC、SQL、Log4J 、ブートストラップ、Jackson、Elastic Beanstalk 設定ファイルの使用方法を示しています。

- Locust Load Generator - このプロジェクトは、Java SE プラットフォームの機能を使用して Locust (Python で書かれた負荷生成ツール) をインストールして実行する方法を示しています。このプロジェクトには、Locust をインストールして定義する設定ファイル、DynamoDB テーブルを設定するビルドスクリプト、Locust を実行する Procfile が含まれています。
- Share Your Thoughts (チュートリアル) - Amazon RDS、Composer、設定ファイルでの MySQL の使用方法を示す PHP アプリケーション
- 新しいスタートアップ (チュートリアル) - DynamoDB、 AWS SDK for JavaScript in Node.js、npm パッケージ管理、設定ファイルの使用方法を示す Node.js サンプルアプリケーション。

IIS アプリケーションを Elastic Beanstalk に移行する

AWS Elastic Beanstalk は、Internet Information Services (IIS) で実行されている Windows アプリケーションの効率的な移行パスを提供します。この章で説明する移行機能により、通常クラウド移行に関連する時間と複雑さが大幅に軽減され、移行中にアプリケーションの機能と設定の整合性を維持できます AWS。

eb migrate オペレーション

Elastic Beanstalk コマンドラインインターフェイス (EB CLI) eb migrateの コマンドを使用して、IIS アプリケーションを自動的に検出、パッケージ化、および にデプロイします AWS クラウド。このプロセスでは、アプリケーションの機能を維持し、バインディング、アプリケーションプール、認証設定などの設定を保持します。

次の手順は、アプリケーションを に移行するために `eb migrate`オペレーションが実行するプロセスをまとめたものです AWS クラウド。

1. IIS サイトとその設定を検出します。
2. アプリケーションのコンテンツと設定をパッケージ化します。
3. Elastic Beanstalk 環境とアプリケーションを作成します。
4. 保存された設定でアプリケーションをデプロイします。

ワークフローと場所の実行オプション

eb migrate コマンドは、柔軟な移行ワークフローと実行場所のオプションを提供します。デフォルトでは、Elastic Beanstalk に移行するアプリケーションを含むターゲットサーバーで コマンドを実行します。アプリケーションサーバーでコマンドを直接実行できない場合は、 `remote`オプションを使用して、アプリケーションと設定を含むターゲットサーバーに接続する踏み台ホストからコマンドを実行します。2 つのステップで移行を完了するには、 `archive-only`オプションを使用してデプロイせずに移行パッケージを生成し、後で `archive`オプションを使用してデプロイすることもできます。

eb migrate コマンドのリファレンス情報については、「」を参照してくださいthe section called “eb migrate”。

トピック

以下のトピックでは、IIS アプリケーションを Elastic Beanstalk に移行する方法について詳しく説明します。

- the section called “前提条件” - Windows アプリケーションを AWS Elastic Beanstalk 環境に移行するために必要なソフトウェア、アクセス、アクセス許可を理解します。
- the section called “移行用語集” - IIS コンポーネントが Elastic Beanstalk リソースにどのようにマッピングされるかを理解する
- the section called “IIS から Elastic Beanstalk への移行マッピング” - IIS コンポーネントが Elastic Beanstalk リソースにどのようにマッピングされるかを理解する
- the section called “IIS の基本的な移行” - 基本的な移行を実行する方法について説明します。
- the section called “高度な移行シナリオ” - 複雑な移行シナリオを処理する
- the section called “セキュリティ設定” - 移行中にセキュリティ設定を構成する
- the section called “ネットワーク構成” - ネットワークとポートの設定を管理する
- the section called “トラブルシューティングと診断” - 一般的な移行問題のトラブルシューティング
- the section called “移行オプション: EB CLI と MGN” - 2 つの主要な移行オプションを比較します。

前提条件

eb migrate コマンドを使用する前に、環境が次の要件を満たしていることを確認してください。

IIS のインストールとバージョン

移行元のサーバーは、Internet Information Services (IIS) バージョン 7.0 以降を実行する必要があります。Windows Server 2016 以降の IIS 10.0 は、移行に最も互換性のある環境を提供します。

IIS バージョンを確認するには、次のコマンドを実行します。

```
PS C:\migrations_workspace> Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\InetStp\"
...
SetupString             : IIS 10.0
VersionString           : Version 10.0
...
```

Windows Server の要件

最適な互換性を確保するために、ソース環境は Windows Server 2016 以降を実行する必要があります。Elastic Beanstalk は、ターゲットプラットフォームとして次の Windows Server バージョンをサポートしています。

- Windows Server 2025
- Windows Server 2022
- Windows Server 2019
- Windows Server 2016

EB CLI のインストール

- デフォルトのワークフロー (`--remote`オプションなし) :
  - Python と Elastic Beanstalk コマンドラインインターフェイス (EB CLI) は、Elastic Beanstalk に移行するアプリケーションを含むサーバーにインストールする必要があります。必須ではありませんが、「」で説明されているように、EB CLI を`virtualenv`サンドボックス内にインストールすることをお勧めしますthe section called “virtualenv に をインストールする”。
- `--remote` オプションの使用 :
  - Python と Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を踏み台ホストにインストールする必要があります。必須ではありませんが、「」で説明されているように、EB CLI を`virtualenv`サンドボックス内にインストールすることをお勧めしますthe section called “virtualenv に をインストールする”。

必要な アクセス許可

次の認証情報とアクセス許可が必要です。

- ソース IIS サーバーまたは踏み台ホストに対する管理者権限 (`--remote`オプションを使用している場合) 。
- AWS Elastic Beanstalk リソースを作成および管理するためのアクセス許可を持つ 認証情報

ウェブデプロイ 3.6

Microsoft Web Deploy ツール (バージョン 3.6 以降) は、ソースサーバーまたは踏み台ホスト (`--remote`オプションを使用している場合) にインストールする必要があります。このツールは、アプリケーションをパッケージ化eb migrateするために で使用されます。

インストールを確認するには、次のコマンドを実行します。

:

```
PS C:\migrations_workspace> Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\IIS
 Extensions\MSDeploy\3" -Name InstallPath

InstallPath  : C:\Program Files\IIS\Microsoft Web Deploy V3\
```

```
...
```

インストール手順については、Microsoft Windows 製品ドキュメントウェブサイトの「IIS 8.0 以降でのウェブデプロイのインストールと設定」を参照してください。

ネットワーク要件

- デフォルトのワークフロー (`--remote`オプションなし) :
  - ソースサーバーには、 AWS サービスへのアウトバウンドインターネットアクセスが必要です。
- `--remote` オプションの使用 :
  - ソースサーバーには、 AWS サービスへのアウトバウンドインターネットアクセスが必要です。
  - 踏み台ホストからの発信ネットワーク接続とリモートマシンへの着信接続を許可する適切なセキュリティグループの進入ルールを設定します。リモートマシンにアクセスするために、踏み台ホストの IP がポート 22 の TCP 経由で許可リストに登録されていることを確認します。
  - SSH クライアントがリモートマシンと踏み台ホストにインストールされ、実行されていることを確認します。
  - ファイアウォール設定に、ポート 22 を開くか、クライアントへの接続を許可する適切なルールが含まれていることを確認します。
  - 移行を試みる前に、踏み台ホストからリモートホストに手動で SSH 接続をテストします。

移行用語集

この用語集では、IIS、Elastic Beanstalk、および IIS アプリケーションの Elastic Beanstalk への移行に関連する主要な用語と概念の定義を示します。

Windows、IIS、および .NET の用語

IIS

Microsoft が Windows Server 用に開発したウェブサーバーソフトウェアである Internet Information Services。IIS はウェブサイト、ウェブアプリケーション、ウェブサービスをホストし、ASP.NET やその他のウェブテクノロジーを実行するためのプラットフォームを提供します。Elastic Beanstalk への移行中、IIS サイトとその設定はパッケージ化され、 AWS クラウドの Windows Server インスタンスにデプロイされます。

IIS バージョン 7.0 以降は移行がサポートされており、Windows Server 2016 以降では IIS 10.0 が最も互換性のある環境を提供します。

.NET Framework

Windows アプリケーションの構築と実行のために Microsoft によって開発されたソフトウェア開発プラットフォーム。フレームワーククラスライブラリ (FCL) と呼ばれる大規模なクラスライブラリを提供し、複数のプログラミング言語間の言語相互運用性をサポートしています。

Elastic Beanstalk に移行する場合、.NET Framework 上に構築されたアプリケーションは、クラウド環境の同じバージョンのフレームワークで引き続き実行されます。Elastic Beanstalk は、Windows Server プラットフォームで複数の .NET Framework バージョン (4.x) をサポートしています。

.NET Core

.NET Framework のクロスプラットフォーム、オープンソースの後継であり、よりモジュール的で軽量な設計になっています。.NET Core (現在は .NET 5 以降と呼ばれています) を使用すると、開発者は Windows、Linux、macOS で実行されるアプリケーションを構築できます。

.NET Core 上に構築されたアプリケーションを Elastic Beanstalk に移行する場合、アプリケーションの要件と依存関係に応じて、Windows Server プラットフォームまたは Linux ベースのプラットフォームを選択できます。

共通言語ランタイム (CLR)

.NET プログラムの実行を管理する .NET Framework の仮想マシンコンポーネント。CLR は、メモリ管理、型安全、例外処理、ガベージコレクション、スレッド管理などのサービスを提供します。

Elastic Beanstalk に移行すると、選択した Windows Server プラットフォームで適切な CLR バージョンが自動的に利用可能になり、アプリケーションの要件との互換性が確保されます。

サイト

ウェブアプリケーションまたはサービスを表す IIS の論理コンテナ。IP アドレス、ポート、ホストヘッダーの一意のバインディングによって識別されます。各 IIS サイトには独自のアプリケーションプール、バインディング、および設定があり、1 つ以上のアプリケーションを含めることができます。

アプリケーション

特定の URL スペースのリクエストを処理する IIS サイト内のコンテンツとコードファイルのグループ化。IIS のアプリケーションは独自の設定を持つことができ、通常は`web.config`ファイルに保存されます。

Elastic Beanstalk に移行する場合、アプリケーションはパス構造と構成設定で保持されます。移行プロセスにより、ネストされたアプリケーションがクラウド環境で階層と URL パスを維持できるようになります。

ApplicationPool

セキュリティ、信頼性、パフォーマンス管理を向上させるためにウェブアプリケーションを分離する IIS 機能。アプリケーションプールは、.NET Framework バージョン、パイプラインモード、アイデンティティ設定など、アプリケーションのランタイム環境を定義します。

VirtualDirectory

サイトのルートディレクトリ外の場所からコンテンツを提供できるようにする IIS のディレクトリマッピング。仮想ディレクトリを使用すると、統合された URL 構造をユーザーに提示しながら、さまざまな物理的な場所にコンテンツを整理できます。

Elastic Beanstalk に移行する場合、仮想ディレクトリはパスマッピングで保持されます。eb migrate コマンドは、同じ URL パスを維持するために、必要なディレクトリ構造と設定をクラウド環境に作成します。

ARR

アプリケーションリクエストルーティング。ウェブサーバーにロードバランシングとプロキシ機能を提供する IIS 拡張機能です。ARR は、複数のサーバー間で URL ベースのルーティング、HTTP リクエスト転送、ロード分散を有効にします。

Elastic Beanstalk への移行中、ARR 設定は EC2 インスタンスへの ARR 機能のインストールと適切なルーティングルールの設定によって保持されます。複雑なルーティングシナリオでは、移行プロセスで Application Load Balancer ルールを活用して同様の機能を実装することもできます。

URL の書き換え

ウェブアプリケーションに到達する前に、定義されたルールに基づいてリクエストされた URLs を変更する IIS モジュール。URL の書き換えにより、パターンと条件に基づいて URL の操作、リダイレクト、コンテンツ配信が可能になります。

Elastic Beanstalk に移行する場合、`web.config`ファイルからの URL 書き換えルールは、可能な場合は ALB ルーティングルールに変換されるか、EC2 インスタンスの IIS 設定に保持されます。これにより、URL パターンとリダイレクトがクラウド環境で期待どおりに機能し続けます。

msdeploy.exe

ウェブアプリケーションとウェブサイトを IIS サーバーにデプロイするために使用されるコマンドラインツール。Web Deploy とも呼ばれ、ウェブアプリケーション、ウェブサイト、サーバー設定をパッケージ化、同期、デプロイする方法を提供します。

eb migrate コマンドは、ウェブデプロイ (バージョン 3.6 以降) を使用して、Elastic Beanstalk への移行中にアプリケーションをパッケージ化します。移行プロセスが正しく機能するには、このツールをソースサーバーにインストールする必要があります。

物理パス

IIS サイトまたはアプリケーションのコンテンツファイルが保存される実際のファイルシステムの場所。物理パスは、IIS サーバーにアクセスできるローカルディレクトリ、ネットワーク共有、またはその他のストレージロケーションを指すことができます。

Elastic Beanstalk への移行中、物理パスは環境内の EC2 インスタンスの適切な場所にマッピングされます。移行プロセスは、すべてのファイルがクラウド環境に適切にデプロイされるようにしながら、コンテンツ構造を保持します。

applicationHost.config

サーバー全体の設定を定義し、すべてのサイト、アプリケーション、仮想ディレクトリの設定を含む IIS のルート設定ファイル。このファイルは `%windir%\System32\inetsrv\config` ディレクトリにあり、IIS サーバーの全体的な動作を制御します。

Elastic Beanstalk に移行すると、 からの関連設定`applicationHost.config`が抽出され、環境内の EC2 インスタンスの IIS 設定に適用されます。これにより、移行中にサーバー全体の設定が保持されます。

web.config

アプリケーションまたはディレクトリレベルでアプリケーション設定、セキュリティ、動作を制御するために ASP.NET アプリケーションで使用される XML ベースの設定ファイル。`web.config` ファイルには、認証、認可、セッション状態、コンパイル、カスタムアプリケーションパラメータの設定を含めることができます。

Elastic Beanstalk への移行中、`web.config`ファイルは保持され、アプリケーションでデプロイされます。移行プロセスにより、アプリケーション固有の設定がクラウド環境で期待どおりに機能し続けます。

DefaultDocument

ユーザーがファイル名を指定せずにディレクトリをリクエストしたときに処理するデフォルトファイルを指定する IIS 機能。デフォルトドキュメントはデフォルトで有効になっており、IIS 7 では`applicationHost.config`、ファイル内のデフォルトドキュメントファイルをサーバー全体のデフォルトとして定義します。Default.htm、Default.asp、Index.htm、Index.html、lisstart.htm。

Elastic Beanstalk に移行する場合、デフォルトのドキュメント設定は EC2 インスタンスの IIS 設定に保持されるため、ディレクトリリクエストはクラウド環境で一貫して処理されます。

system.webServer

モジュール、ハンドラー、その他のサーバー動作の IIS 固有の設定`applicationHost.config`を含む `web.config`または の設定セクション。このセクションでは、IIS がリクエストを処理し、モジュールを管理し、サーバー機能を設定する方法を制御します。

Elastic Beanstalk への移行中、system.webServer 設定はアプリケーションの`web.config`ファイルに保持され、環境内の EC2 インスタンスへの IIS インストールに適用されます。これにより、IIS 固有の動作がクラウド環境で維持されます。

Elastic Beanstalk 用語

プラットフォーム

オペレーティングシステム、プログラミング言語ランタイム、ウェブサーバー、アプリケーションサーバー、およびアプリケーションを実行するためのソフトウェアスタックを定義する Elastic Beanstalk コンポーネントの組み合わせ。

Windows 移行の場合、Elastic Beanstalk は Windows Server 2016、2019、2022 に基づくプラットフォームと IIS およびさまざまな .NET Framework バージョンを提供し、ソース環境との互換性を確保します。

SolutionStack

アプリケーションの実行に必要なオペレーティングシステム、ランタイム、およびその他のコンポーネントを指定する、Elastic Beanstalk の事前定義されたプラットフォーム設定。概念的にはプラットフォームと同じで、環境を運用するために同じ意味で使用されました。

移行中、 eb migrate コマンドはソース環境の設定に基づいて適切なソリューションスタックを選択し、IIS アプリケーションとの互換性を確保します。

CreateEnvironment

アプリケーションバージョンをホストする新しい環境を作成する Elastic Beanstalk API アクション。この API は、 eb migrate コマンドによって、移行されたアプリケーションに必要な AWS リソースをプロビジョニングするために使用されます。

移行プロセスでは、インスタンスタイプ、環境変数、オプション設定など、ソース IIS 環境に基づいて適切な環境パラメータを設定します。

CreateApplicationVersion

Amazon S3 に保存されているソースバンドルから新しいアプリケーションバージョンを作成する Elastic Beanstalk API アクション。eb migrate コマンドはこの API を使用して、パッケージ化された IIS アプリケーションを Elastic Beanstalk のバージョンとして登録します。

移行中、アプリケーションファイルと設定はパッケージ化され、Amazon S3 にアップロードされ、デプロイ前にアプリケーションバージョンとして登録されます。

DescribeEvents

デプロイ、設定変更、運用上の問題など、環境のイベントのリストを取得する Elastic Beanstalk API アクション。eb migrate コマンドは、この API を使用して移行の進行状況をモニタリングします。

移行後に eb events コマンドを使用して、環境のイベント履歴を表示することもできます。

DescribeEnvironmentHealth

環境のインスタンスやその他のコンポーネントに関する詳細なヘルス情報を提供する Elastic Beanstalk API アクション。この API は、デプロイ後に移行されたアプリケーションの正常性を検証するために使用されます。

移行後、 eb health コマンドを使用して環境のステータスを確認し、注意が必要な問題を特定できます。

HealthD

Elastic Beanstalk のモニタリングエージェント。環境内の EC2 インスタンスのメトリクスの収集、ログのモニタリング、ヘルスステータスの報告を行います。HealthD は、移行されたアプリケーションの拡張ヘルスレポートを提供します。

移行後、HealthD はアプリケーションのパフォーマンス、リソース使用率、リクエストの成功率をモニタリングし、環境の状態を包括的に把握します。

バンドルログ

EC2 インスタンスから Amazon S3 にログを圧縮してアップロードし、一元化されたストレージと分析を行う Elastic Beanstalk の機能。この機能は、移行されたアプリケーションに関する問題のトラブルシューティングに役立ちます。

移行後、 eb logs コマンドを使用して環境からログを取得して表示できます。

aws-windows-deployment-manifest.json

ソフトウェアパッケージまたはアプリケーションの内容、依存関係、および設定を記述するファイル。このマニフェストは、移行プロセス中に生成され、IIS アプリケーションを Elastic Beanstalk にデプロイする方法を定義します。

カスタムマニフェストセクション

アプリケーションデプロイのカスタム制御`aws-windows-deployment-manifest.json`を提供する 内のセクション。このセクションでは、デプロイプロセス中に実行される PowerShell スクリプトとコマンドについて説明します。

移行中、カスタムマニフェストセクションが生成され、仮想ディレクトリの設定、アクセス許可管理、アプリケーションプールの設定など、IIS 設定の特定の側面が処理されます。

EB CLI

Elastic Beanstalk アプリケーションと環境を作成、設定、管理するためのコマンドを提供するコマンドラインツール。EB CLI には、IIS アプリケーションを Elastic Beanstalk に移行するためのeb migrateコマンドが含まれています。

移行後も、EB CLI を引き続き使用して、環境の管理、更新のデプロイ、ヘルスのモニタリング、その他の管理タスクを実行できます。

オプション設定

Elastic Beanstalk が環境内の AWS リソースをプロビジョニングおよび設定する方法を定義する設定値。オプション設定は、ロードバランサー、インスタンス、環境プロセスなど、環境のさまざまなコンポーネントを表す名前空間に編成されます。

移行中、 eb migrate コマンドは IIS 設定に基づいて適切なオプション設定を生成し、クラウド環境がソース環境の機能と一致するようにします。詳細については、「Elastic Beanstalk デベロッパーガイド」の「設定オプション」を参照してください。

aws:elbv2:listener:default

Application Load Balancer のデフォルトリスナーの Elastic Beanstalk 設定名前空間。移行中、この名前空間は IIS サイトバインディングに基づいて設定され、適切なトラフィックルーティングが確保されます。

デフォルトのリスナーは通常、ポート 80 で HTTP トラフィックを処理し、ルーティングルールに従ってアプリケーションインスタンスに転送されます。

aws:elbv2:listener:listener_port

Application Load Balancer 上の特定のリスナーポートの Elastic Beanstalk 設定名前空間。この名前空間は、ポート 443 の HTTPS など、移行されたアプリケーションの追加のリスナーを設定するために使用されます。

移行中、リスナーは IIS サイトのポートバインディングに基づいて作成され、ソース環境と同じポートでアプリケーションにアクセスできるようにします。

aws:elbv2:listenerrule:rule_name

Application Load Balancer リスナーのルーティングルールを定義するための Elastic Beanstalk 設定名前空間。これらのルールは、パスパターンまたはホストヘッダーに基づいて、受信リクエストを異なるターゲットグループにルーティングする方法を決定します。

移行中、IIS アプリケーションの URL 構造に一致するリスナールールが作成され、リクエストが正しいアプリケーションパスにルーティングされます。

aws:elasticbeanstalk:environment:process:default

環境のデフォルトプロセスの Elastic Beanstalk 設定名前空間。この名前空間は、ヘルスチェック設定、ポートマッピング、プロキシ設定など、デフォルトのウェブアプリケーションプロセスの設定方法を定義します。

移行中、デフォルトのプロセスはプライマリ IIS サイトの設定に基づいて設定され、適切なヘルスモニタリングとリクエスト処理が保証されます。

aws:elasticbeanstalk:environment:process:process_name

環境内の特定の名前付きプロセスの Elastic Beanstalk 設定名前空間。この名前空間を使用すると、IIS に複数のアプリケーションプールがある場合と同様に、異なる設定で複数のプロセスを定義できます。

移行中に、ソース環境とは異なるサイトバインディングを表す追加のプロセスを作成する場合があります。

> **Note**
>
> このトピックで説明されている用語の一部の詳細については、以下のリソースを参照してください。
>
> - Elastic Beanstalk API アクション - AWS Elastic Beanstalk API リファレンス
> - サポートされているプラットフォームバージョンを含む Elastic Beanstalk プラットフォーム - プラットフォームガイドの「サポートされているAWS Elastic Beanstalk プラットフォーム」
> - このガイドすべての環境に対する汎用オプションの Elastic Beanstalk 設定名前空間 -
> - このガイドElastic Beanstalk を管理するための EB コマンドラインインターフェイス (EB CLI) の設定の EB CLI または特定の EB CLI コマンド -

Python 用語

pip

Python のパッケージインストーラ。Python で記述されたソフトウェアパッケージのインストールと管理に使用されます。EB CLI は pip を使用してインストールおよび更新されます。

移行プロセス中、pip を使用して EB CLI パッケージとその依存関係をソースサーバーにインストールし、移行に必要なツールを提供します。

PyPI

Python Package Index は、pip がパッケージを取得してインストールするサードパーティーの Python ソフトウェアパッケージの公式リポジトリです。EB CLI とその依存関係は PyPI でホストされます。

移行用に EB CLI をインストールすると、pip は PyPI に接続して必要なパッケージをダウンロードしてインストールします。

virtualenv

分離された Python 環境を作成するためのツール。これにより、さまざまなプロジェクトが競合することなく独自の依存関係とパッケージを持つことができます。他の Python アプリケーションとの競合を避けるため、EB CLI をインストールするときは virtualenv を使用することをお勧めします。

EB CLI をインストールする前に仮想環境を作成すると、移行ツールに適切な依存関係を持つクリーンで分離された環境が確保されます。

pywin32

多くの Windows APIs へのアクセスを提供する Python 拡張機能のセット。Windows オペレーティングシステムとそのコンポーネントとのやり取りを可能にします。EB CLI は pywin32 を使用して、移行中に Windows 固有の機能を操作します。

移行プロセス中、pywin32 は IIS 設定、Windows レジストリ設定、およびアプリケーションを適切にパッケージ化して移行するために必要なその他のシステム情報にアクセスするために使用されます。

Pythonnet

Python コードが .NET Framework および .NET Core アプリケーションとやり取りできるようにするパッケージ。この統合により、EB CLI は移行プロセス中に .NET コンポーネントを操作できます。

移行プロセスでは、アプリケーションを分析およびパッケージ化して Elastic Beanstalk にデプロイする際に、Pythonnet を使用して .NET アセンブリおよびコンポーネントとやり取りする場合があります。

基本的な IIS 移行の実行

このセクションでは、 eb migrate コマンドを使用して IIS アプリケーションを Elastic Beanstalk に移行するプロセスについて説明します。

IIS 環境の探索

変更を行う前に、サーバーに存在するリソースを理解する必要があります。まずeb migrate explore、次の例に示すように、 を実行して IIS サイトを調べます。

```
PS C:\migrations_workspace> eb migrate explore
```

このコマンドは IIS サイトを表示します。次のリストを参照してください。

```
Default Web Site
Intranet
API.Internal
Reports
```

バインディング、アプリケーション、仮想ディレクトリなど、各サイトの設定の詳細を表示するには、次の例に示すように、 `--verbose`オプションを追加します。

```
PS C:\migrations_workspace> eb migrate explore --verbose
```

次のリストは、 コマンドが提供する環境に関する包括的な情報を示しています。

```
1: Default Web Site:
  - Bindings:
    - *:80:www.example.com
    - *:443:www.example.com
  - Application '/':
    - Application Pool: DefaultAppPool
    - Enabled Protocols: http
    - Virtual Directories:
      - /:
        - Physical Path: C:\inetpub\wwwroot
        - Logon Method: ClearText
  - Application '/api':
    - Application Pool: ApiPool
    - Enabled Protocols: http
```

```
    - Virtual Directories:
      - /:
        - Physical Path: C:\websites\api
        - Logon Method: ClearText
2: Intranet:
...
3. API.Internal:
...
4. Reports:
...
```

検出出力について

詳細な出力は、移行計画に以下の重要な情報を提供します。

サイト

検出出力には、サーバー上のすべての IIS サイトが一覧表示されます。各サイトは名前 (「デフォルトウェブサイト」、「イントラネット」、「API.内部」など) で識別され、順番に番号が付けられます。サーバーに複数のサイトが存在する場合、 `eb migrate` コマンドは移行戦略に応じて、それぞれを個別に、またはまとめてパッケージ化してデプロイできます。

バインディング

プロトコルバインディングは、サイトがどのプロトコル (HTTP/HTTPS) を使用し、どのポートで動作するかを明らかにします。バインディング情報には、ドメインベースのルーティング設定を定義するホストヘッダー要件が含まれます。

アプリケーション

アプリケーションパスは、IIS 設定内のルートアプリケーション構造とネストされたアプリケーション構造の両方を表示します。アプリケーションプールの割り当ては、セキュリティとリソース管理のためにアプリケーションがどのように分離されているかを示します。

仮想ディレクトリ

物理パスマッピングは、コンテンツがファイルシステム上に存在する場所を示します。認証設定には、移行後に維持する必要がある特別なアクセス要件が表示されます。

移行の準備

環境を理解し、サーバーが前提条件を満たしていることを確認します。まず、次の PowerShell コマンドを使用して IIS バージョンを確認します。

```
PS C:\migrations_workspace> Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\InetStp\" -Name
 MajorVersion
```

IIS 7.0 以降が必要です。移行ツールは、Web Deploy 3.6 を使用してアプリケーションをパッケージ化します。次のコマンドを使用してインストールを確認します。

```
PS C:\migrations_workspace> Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\IIS Extensions
\MSDeploy\3" -Name InstallPath
```

Web Deploy がサーバーにインストールされていない場合は、Microsoft Web Platform Installer ダウンロードページからダウンロードできます。

最初の移行

デフォルトウェブサイトの基本的な移行から始めましょう。次の例は、最も単純なコマンド を示していますeb migrate。

```
PS C:\migrations_workspace> eb migrate
```

このコマンドは、次の出力例に示すように、一連の自動ステップを開始します。

```
Identifying VPC configuration of this EC2 instance (i-0123456789abcdef0)
  id: vpc-1234567890abcdef0
  publicip: true
  elbscheme: public
  ec2subnets: subnet-123,subnet-456,subnet-789
  securitygroups: sg-123,sg-456
  elbsubnets: subnet-123,subnet-456,subnet-789

Using .\migrations\latest to contain artifacts for this migration run.
Generating source bundle for sites, applications, and virtual directories...
  Default Web Site/ -> .\migrations\latest\upload_target\DefaultWebSite.zip
```

移行ツールは、デプロイアーティファクトを含む構造化ディレクトリを作成します。次のリストは、ディレクトリ構造を示しています。

```
C:\migration_workspace\
### .\migrations\latest\
    ### upload_target\
```

```
        ### DefaultWebSite.zip
        ### aws-windows-deployment-manifest.json
        ### ebmigrateScripts\
            ### site_installer.ps1
            ### permission_handler.ps1
            ### >other helper scripts<
```

移行の制御

移行プロセスをより詳細に制御するには、次のコマンドを使用して、移行するサイトを正確に指定できます。

```
PS C:\migrations_workspace> eb migrate --sites "Default Web Site,Intranet"
```

次のコマンド例に示すように、環境名とアプリケーション名をカスタマイズすることもできます。

```
PS C:\migrations_workspace> eb migrate `
    --sites "Default Web Site" `
    --application-name "CorporateApp" `
    --environment-name "Production"
```

オプションの完全なリストについては、「」を参照してくださいthe section called “eb migrate”。

進行状況の監視

移行中、 eb migrateはリアルタイムのステータス更新を提供します。次の出力例を参照してください。

```
...
Creating application version
Creating environment... This may take a few minutes

2024-03-18 18:12:15    INFO    Environment details for: Production
  Application name: CorporateApp
  Region: us-west-2
  Deployed Version: app-230320_153045
  Environment ID: e-abcdef1234
  Platform: 64bit Windows Server 2019 v2.7.0 running IIS 10.0
  Tier: WebServer-Standard-1.0
  CNAME: production.us-west-2.elasticbeanstalk.com
  Updated: 2024-03-20 15:30:45
```

```
2025-03-18 18:12:17    INFO    createEnvironment is starting.
2025-03-18 18:12:19    INFO    Using elasticbeanstalk-us-east-1-180301529717 as Amazon
 S3 storage bucket for environment data.
2025-03-18 18:12:40    INFO    Created security group named: sg-0fdd4d696a26b086a
2025-03-18 18:12:48    INFO    Environment health has transitioned to Pending.
 Initialization in progress (running for 7 seconds). There are no instances.
...
2025-03-18 18:23:59    INFO    Application available at EBMigratedEnv-arrreal3.us-
east-1.elasticbeanstalk.com.
2025-03-18 18:24:00    INFO    Successfully launched environment: EBMigratedEnv-
arrreal3
```

移行の検証

環境の準備が完了すると、Elastic Beanstalk はデプロイを検証するいくつかの方法を提供します。

アプリケーションにアクセスする

ウェブブラウザでアプリケーション URL (CNAME) を開き、正しく動作していることを確認します。

環境の状態を確認する

eb health コマンドを使用して、環境の状態を表示します。

```
PS C:\migrations_workspace> eb health
```

次の画面イメージは、インスタンスの状態、アプリケーションレスポンスメトリクス、およびシステムリソース使用率を示しています。

eb logs コマンドを使用してログにアクセスし、問題をトラブルシューティングします。

```
PS C:\migrations_workspace> eb logs --zip
```

eb logs コマンドは、 `.elasticbeanstalk/logs` ディレクトリにログをダウンロードします。詳細については、「the section called “CloudWatch Logs”」を参照してください。

インスタンスに接続する

移行中にキーペアを指定した場合は、RDP を使用してインスタンスに接続し、直接トラブルシューティングを行うことができます。

Elastic Beanstalk コンソールにアクセスする

環境のヘルス、ログ、設定プロパティは、その環境の環境マネジメントコンソールから表示できます。

移行アーティファクトの管理

eb migrate コマンドは、移行プロセス中にローカルアーティファクトを作成します。これらのアーティファクトには機密情報が含まれており、時間の経過とともに大量のディスク容量を消費する可能性があります。次の例に示すように、 cleanupサブコマンドを使用してこれらのアーティファクトを管理します。

```
PS C:\migrations_workspace> eb migrate cleanup
Are you sure you would like to cleanup older artifacts within ./migrations/? (Y/N):
```

確認なしで強制クリーンアップするには、 --forceオプションを使用します。

```
PS C:\migrations_workspace> eb migrate cleanup --force
```

クリーンアッププロセスでは、最後に成功した移行が `./migrations/latest` ディレクトリに保存され、古い移行ディレクトリが削除されます。

ネットワーク設定とポート設定

このセクションでは、VPC 設定、ポート設定、マルチサイトデプロイなど、IIS 移行のネットワーク設定オプションについて説明します。

VPC の構成

eb migrate コマンドは、Elastic Beanstalk 環境に柔軟な VPC 設定オプションを提供します。このツールは、ソース EC2 インスタンスから VPC 設定を検出するか、コマンドラインパラメータを使用してカスタム VPC 設定を受け入れることができます。VPC で Elastic Beanstalk を設定する方法については、Amazon VPC で Elastic Beanstalk を使用する「」を参照してください。

自動 VPC 検出

EC2 インスタンスで eb migrateを実行すると、ソース環境の EC2 インスタンスから VPC 設定を自動的に検出して使用します。次の出力例は、検出した設定情報を示しています。

```
PS C:\migrations_workspace > eb migrate
Identifying VPC configuration of this EC2 instance (i-0123456789abcdef0):
  id: vpc-1234567890abcdef0
  publicip: true
  elbscheme: public
  ec2subnets: subnet-123,subnet-456,subnet-789
  securitygroups: sg-123,sg-456
  elbsubnets: subnet-123,subnet-456,subnet-789
...
```

検出された設定には以下が含まれます。

- VPC 識別子
- パブリック IP 割り当て設定
- ロードバランサースキーム (パブリック/プライベート)
- EC2 インスタンスサブネットの割り当て
- セキュリティグループの関連付け
- ロードバランサーサブネットの割り当て

オンプレミスホストまたは非AWS クラウドホスト

がオンプレミスサーバーまたはAWS クラウド以外のホストからeb migrate実行されると、Elastic Beanstalk サービスは AWS アカウントのデフォルト VPC を使用します。次のリストは、コマンドと出力の例を示しています。

```
PS C:\migrations_worspace> eb migrate `
      -k windows-test-pem `
      --region us-east-1 `
      -a EBMigratedEnv `
      -e EBMigratedEnv-test2 `
      --copy-firewall-config
Determining EB platform based on host machine properties
Using .\migrations\latest to contain artifacts for this migration run.
...
```

Elastic Beanstalk が環境のデフォルト VPC を設定するAmazon VPC で Elastic Beanstalk を使用する方法については、「」を参照してください。

カスタム VPC 設定

特定の VPC 設定が必要なソース環境 (EC2、オンプレミス、またはAWS クラウド以外) の場合は、次の例のような VPC 設定ファイルを指定します。

```
{
    "id": "vpc-12345678",
    "publicip": "true",
    "elbscheme": "public",
    "ec2subnets": ["subnet-a1b2c3d4", "subnet-e5f6g7h8"],
    "securitygroups": "sg-123456,sg-789012",
    "elbsubnets": ["subnet-a1b2c3d4", "subnet-e5f6g7h8"]
}
```

次のコマンドを使用して、この設定を適用します。

```
PS C:\migrations_workspace> eb migrate --vpc-config vpc-config.json
```

> **Note**
>
> VPC 設定ファイルには、VPC ID を指定する `id`フィールドが必要です。他のすべてのフィールドはオプションであり、Elastic Beanstalk は指定しないフィールドのデフォルト値を使用します。

> **Important**
>
> パラメータを指定すると、移行はソース環境からの既存の VPC 設定を無視します`--vpc-config`。このパラメータを使用すると、移行では、渡す設定ファイルで指定された VPC 設定のみが使用されます。このパラメータを使用すると、ソースインスタンスの VPC 設定を検出するか、デフォルトの VPC を使用するというデフォルトの動作が上書きされます。

これらのシナリオでは、 `--vpc-config`パラメータを使用します。

- 検出可能な VPC 設定がないnon-EC2 環境を移行する場合
- ソース環境で使用される VPC とは異なる VPC に移行する場合
- サブネット選択またはセキュリティグループ設定をカスタマイズする必要がある場合

- 自動検出で目的の VPC 設定が正しく識別されない場合
- オンプレミスから移行し、デフォルトの VPC を使用しない場合

ネットワークセキュリティ設定

デフォルトでは、 はターゲットインスタンスでポート 80 eb migrateを開きますが、ソースマシンから他の Windows Firewall ルールをコピーしません。すべてのファイアウォール設定を含めるには、次のコマンドを使用します。

```
PS C:\migrations_workspace> eb migrate --copy-firewall-config
```

このコマンドは、次のアクションを実行します。

- IIS サイトバインディングで使用されるポートを識別します
- 対応するファイアウォールルールを取得します。
- ターゲットインスタンスのルールを再作成するための PowerShell スクリプトを生成します
- ソースマシンからのポート 80 の DENY ルールを保持します (それ以外の場合、ポート 80 はデフォルトで許可されます)

ソースマシンに次の例で指定されたファイアウォールルールがあるユースケースを考えてみましょう。

```
# Source machine firewall configuration
Get-NetFirewallRule | Where-Object {$_.Enabled -eq 'True'} | Get-NetFirewallPortFilter
 | Where-Object {$_.LocalPort -eq 80 -or $_.LocalPort -eq 443 -or $_.LocalPort -eq
 8081}
# Output shows rules for ports 80, 443, and 8081
```

移行では、次の設定を含むスクリプト (`modify_firewall_config.ps1`) が作成されます。

```
New-NetFirewallRule -DisplayName "Allow Web Traffic" -Direction Inbound -Action Allow -
Protocol TCP -LocalPort 80,443
New-NetFirewallRule -DisplayName "Allow API Traffic" -Direction Inbound -Action Allow -
Protocol TCP -LocalPort 8081
```

移行ツールは、次のアクションを自動的に実行します。

- すべての IIS サイトバインディングから HTTP/HTTPS ポートを抽出します
- Windows Firewall INetFwPolicy2 インターフェイスを使用してファイアウォールルールを列挙します
- 指定されたポートを明示的に参照するルールのみを含めるようにルールをフィルタリングします
- HTTP サイトバインディングと HTTPS サイトバインディング、および関連するファイアウォールルールのみを処理する
- 表示名、アクション、プロトコル、有効状態などのルールプロパティを保持します
- ファイアウォールルールの個々のポートとポート範囲の両方を処理します
- ファイアウォール設定スクリプトをデプロイマニフェストに追加します

ロードバランサーの設定

Load Balancer の設定は、 `--vpc-config`引数を使用して指定できます。以下の例は、パラメータを示しています。

スキームの選択

パブリックロードバランサースキームとプライベートロードバランサースキームのいずれかを選択します。

```
{
    "id": "vpc-12345678",
    "elbscheme": "private",
    "elbsubnets": ["subnet-private1", "subnet-private2"]
}
```

サブネットディストリビューション

高可用性を実現するには、ロードバランサーサブネットをアベイラビリティーゾーンに分散します。

```
{
    "elbsubnets": [
        "subnet-az1", // Availability Zone 1
        "subnet-az2", // Availability Zone 2
        "subnet-az3"  // Availability Zone 3
    ]
```

```
}
```

> **Note**
>
> Elastic Beanstalk は Application Load Balancer、Network Load Balancer、Classic Load Balancer による環境作成をサポートしていますが、 eb migrate コマンドは Application Load Balancer のみをサポートします。ロードバランサータイプの詳細については、「Elastic Beanstalk 環境のロードバランサー」を参照してください。

ポート設定を使用したマルチサイトデプロイ

eb migrate コマンドは、アプリケーションが依存関係を共有したり、非標準ポートを使用したりする可能性のある複雑なマルチサイト IIS デプロイを処理します。複数のサイトを持つ一般的なエンタープライズセットアップの例を次に示します。

```
<!-- IIS Configuration -->
<sites>
    <site name="Default Web Site" id="1">
        <bindings>
            <binding protocol="http" bindingInformation="*:80:www.example.com" />
        </bindings>
    </site>
    <site name="InternalAPI" id="2">
        <bindings>
            <binding protocol="http" bindingInformation="*:8081:api.internal" />
        </bindings>
    </site>
    <site name="ReportingPortal" id="3">
        <bindings>
            <binding protocol="http" bindingInformation="*:8082:reports.internal" />
        </bindings>
    </site>
</sites>
```

この設定を移行するには、次のコマンドとパラメータの例を使用します。

```
PS C:\migrations_workspace> eb migrate `
    --sites "Default Web Site,InternalAPI,ReportingPortal" `
    --copy-firewall-config `
```

```
    --instance-type "c5.large"
```

eb migrate コマンドは、各サイトの ID と設定を保持するデプロイパッケージを作成します。コマンドは`aws-windows-deployment-manifest.json`、これらのサイトのデプロイ方法を定義する を生成します。次の例は、生成された json ファイルを示しています。

```
{
    "manifestVersion": 1,
    "deployments": {
        "msDeploy": [
            {
                "name": "DefaultWebSite",
                "parameters": {
                    "appBundle": "DefaultWebSite.zip",
                    "iisPath": "/",
                    "iisWebSite": "Default Web Site"
                }
            }
        ],
        "custom": [
            {
                "name": "InternalAPI",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\\install_site_InternalAPI.ps1"
                    },
                    "restart": {
                        "file": "ebmigrateScripts\\restart_site_InternalAPI.ps1"
                    },
                    "uninstall": {
                        "file": "ebmigrateScripts\\uninstall_site_InternalAPI.ps1"
                    }
                }
            },
            {
                "name": "ReportingPortal",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\\install_site_ReportingPortal.ps1"
                    },
                    "restart": {
                        "file": "ebmigrateScripts\\restart_site_ReportingPortal.ps1"
                    },
```

```
                    "uninstall": {
                        "file": "ebmigrateScripts\\uninstall_site_ReportingPortal.ps1"
                    }
                }
            }
        ]
    }
}
```

移行プロセスでは、元のルーティングロジックを維持する次の Application Load Balancer リスナールールが作成されます。

- ポート 80 トラフィックがデフォルトウェブサイトにルーティングされる
- ポート 8081 トラフィックは InternalAPI にルーティングされます
- ポート 8082 トラフィックは ReportingPortal にルーティングされます

共有設定と依存関係

サイトが設定または依存関係を共有すると、 はこれらの関係を適切にeb migrate処理します。複数のサイトが共通の設定を共有する場合は、次の例を参照してください。

```
<!-- Shared configuration in applicationHost.config -->
<location path="Default Web Site">
    <system.webServer>
        <asp enableSessionState="true" />
        <caching enabled="true" enableKernelCache="true" />
    </system.webServer>
</location>
```

移行プロセスは、次のタスクを完了します。

1. サイト間での共有設定を識別します
2. これらの設定を適用するための適切な PowerShell スクリプトを生成します
3. 設定階層と継承を維持します

ベストプラクティス

移行したアプリケーションのネットワーク設定のベストプラクティスに従うことをお勧めします。以下のグループ化は、概要ガイドラインを提供します。

VPC の設計

- AWS VPC 設計のベストプラクティスに従う
- ロードバランサーと EC2 インスタンスに別々のサブネットを使用する
- 適切なルートテーブルと NACLs
- AWS サービスの VPC エンドポイントを検討する

高可用性

- 複数のアベイラビリティゾーンにデプロイする
- ロードバランサーに少なくとも 2 つのサブネットを使用する
- AZs 間での自動スケーリングを設定する
- 適切なヘルスチェックを実装する

セキュリティ

- セキュリティのベストプラクティスに従う
- セキュリティグループをプライマリアクセスコントロールとして使用する
- ネットワークアクセスコントロールリスト (ACLs) を実装してセキュリティを強化する
- VPC フローログのモニタリング

トラブルシューティング

一般的なネットワーク設定の問題には、次の領域が含まれます。次に、環境のネットワーク設定とヘルスに関する詳細情報を取得するためのコマンドの例を示します。

サブネット設定

```
# Verify subnet availability
PS C:\migrations_workspace> aws ec2 describe-subnets --subnet-ids subnet-id

# Check available IP addresses
PS C:\migrations_workspace>aws ec2 describe-subnets --subnet-ids subnet-id --query
 'Subnets[].AvailableIpAddressCount'
```

セキュリティグループアクセス

```
# Verify security group rules
PS C:\migrations_workspace> aws ec2 describe-security-groups --group-ids sg-id
```

```
# Test network connectivity
PS C:\migrations_workspace> aws ec2 describe-network-interfaces --filters
 Name=group-id,Values=sg-id
```

ロードバランサーのヘルス

```
# Check load balancer health
PS C:\migrations_workspace> aws elbv2 describe-target-health --target-group-arn
 arn:aws:elasticloadbalancing:region:account-id:targetgroup/group-name/group-id
```

セキュリティ設定と IAM ロール

eb migrate コマンドは、IAM ロール、インスタンスプロファイル、およびサービスロールを通じて AWS セキュリティ設定を管理します。これらのコンポーネントを理解することで、移行中の適切なアクセスコントロールとセキュリティコンプライアンスが保証されます。

インスタンスプロファイル設定

インスタンスプロファイルは、Elastic Beanstalk が環境内の EC2 インスタンスにアタッチする IAM ロールのコンテナとして機能します。を実行するときにeb migrate、カスタムインスタンスプロファイルを指定できます。

```
PS C:\migrations_workspace> eb migrate --instance-profile "CustomInstanceProfile"
```

インスタンスプロファイルを指定しない場合、 はこれらのアクセス許可を持つデフォルトプロファイルeb migrateを作成します。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:ListBucket"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::elasticbeanstalk-*",
                "arn:aws:s3:::elasticbeanstalk-*/*"
            ]
        }
    ]
}
```

サービスロールの管理

サービスロールを使用すると、Elastic Beanstalk はユーザーに代わって AWS リソースを管理できます。次のコマンドを使用して、移行中にカスタムサービスロールを指定します。

```
PS C:\migrations_workspace> eb migrate --service-role "CustomServiceRole"
```

指定しない場合、 は、Elastic Beanstalk がロールを引き受けることを許可する信頼ポリシーaws-elasticbeanstalk-service-roleを使用して、 という名前のデフォルトのサービスロールeb migrateを作成します。このサービスロールは、Elastic Beanstalk が環境の状態をモニタリングし、マネージドプラットフォームの更新を実行するために不可欠です。サービスロールには 2 つの管理ポリシーが必要です。

- `AWSElasticBeanstalkEnhancedHealth` - Elastic Beanstalk が拡張ヘルスレポートシステムを使用してインスタンスと環境のヘルスをモニタリングできるようにします
- `AWSElasticBeanstalkManagedUpdates` - 新しいプラットフォームバージョンが利用可能になったときに環境リソースを更新するなど、Elastic Beanstalk がマネージドプラットフォームの更新を実行できるようにします。

これらのポリシーでは、サービスロールには以下のアクセス許可があります。

- Auto Scaling グループの作成と管理
- Application Load Balancer の作成と管理
- Amazon CloudWatch にログをアップロードする
- EC2 インスタンスを管理する

サービスロールの詳細については、Elastic Beanstalk デベロッパーガイドElastic Beanstalk サービスロールの「」を参照してください。

セキュリティグループの構成

eb migrate コマンドは、IIS サイトのバインディングに基づいてセキュリティグループを自動的に設定します。たとえば、ソース環境にポート 80、443、および 8081 を使用するサイトがある場合、次の設定結果になります。

```
<site name="Default Web Site">
    <bindings>
        <binding protocol="http" bindingInformation="*:80:" />
        <binding protocol="https" bindingInformation="*:443:" />
    </bindings>
</site>
<site name="InternalAPI">
    <bindings>
        <binding protocol="http" bindingInformation="*:8081:" />
    </bindings>
</site>
```

移行プロセスは、次のアクションを完了します。

- インターネットからのポート 80 および 443 でのインバウンドトラフィックを許可するロードバランサーセキュリティグループを作成します (0.0.0.0/0)
- ロードバランサーからのトラフィックを許可する EC2 セキュリティグループを作成します。
- `--copy-firewall-config` が指定されている場合、追加のポート (8081 など) を設定します

デフォルトでは、Application Load Balancer はインターネットからのパブリックアクセスで設定されます。特定の IP 範囲へのアクセスの制限やプライベートロードバランサーの使用など、この動作をカスタマイズする必要がある場合は、 `--vpc-config`パラメータを使用してデフォルトの VPC およびセキュリティグループ設定を上書きできます。

```
PS C:\migrations_workspace> eb migrate --vpc-config vpc-config.json
```

たとえば、次の`vpc-config.json`設定では、プライベートサブネットにプライベートロードバランサーを作成します。

```
{
    "id": "vpc-12345678",
    "publicip": "false",
```

```
    "elbscheme": "internal",
    "ec2subnets": ["subnet-private1", "subnet-private2"],
    "elbsubnets": ["subnet-private1", "subnet-private2"]
}
```

VPC 設定オプションの詳細については、「」を参照してくださいVPC の構成。

SSL 証明書の統合

HTTPS バインディングを使用してサイトを移行する場合は、 AWS Certificate Manager (ACM) を介して SSL 証明書を統合します。

```
PS C:\migrations_workspace> eb migrate --ssl-certificates
 "arn:aws:acm:region:account:certificate/certificate-id"
```

この設定では、次のアクションが完了します。

- 証明書を Application Load Balancer に関連付けます。
- ロードバランサーでの HTTPS 終了を維持します
- ロードバランサーと EC2 インスタンス間の内部 HTTP 通信を保持します

Windows 認証

Windows 認証を使用するアプリケーションの場合、 は次のようにアプリケーションの の認証設定eb migrateを保持`web.config`します。

```
<configuration>
    <system.webServer>
        <security>
            <authentication>
                <windowsAuthentication enabled="true">
                    <providers>
                        <add value="Negotiate" />
                        <add value="NTLM" />
                    </providers>
                </windowsAuthentication>
            </authentication>
        </security>
    </system.webServer>
```

```
</configuration>
```

> **Important**
>
> eb migrate コマンドは、ソース環境からターゲットの Elastic Beanstalk インスタンスにユーザープロファイルまたはアカウントをコピーしません。ソースサーバーで作成したカスタムユーザーアカウントまたはグループは、移行後にターゲット環境で再作成する必要があります。

などの組み込み Windows アカウント`IUSR`、 などのグループ`IIS_IUSRS`、および他のすべての組み込みアカウントとグループは、ターゲット Windows Server インスタンスにデフォルトで含まれます。組み込み IIS アカウントとグループの詳細については、Microsoft ドキュメントの「Understanding Built-In User and Group Accounts in IIS」を参照してください。

アプリケーションがカスタム Windows ユーザーアカウントまたは Active Directory 統合に依存している場合は、移行の完了後にこれらの側面を個別に設定する必要があります。

ベストプラクティスとトラブルシューティング

ロールの管理

Elastic Beanstalk AWS 環境のロールを管理するときに IAM ベストプラクティスを実装します。

ロールの作成と管理

- 可能な場合は、 AWS 管理ポリシーを使用してロールを作成する
- IAM セキュリティのベストプラクティスに従う
- カスタムAWS ポリシーに Policy Generator を使用する
- セキュリティを強化するためのアクセス許可の境界を実装する

モニタリングと監査

ロールの使用状況をモニタリング AWS CloudTrail するには、 を有効にします。

- AWS CloudTrail ユーザーガイドに従う
- リアルタイムモニタリング用に CloudWatch Logs 統合を設定する
- 不正な API コールのアラートを設定する

定期的なレビュープロセス

四半期ごとのレビューサイクルを確立して、次のタスクを実行します。

- IAM Access Analyzer を使用して未使用のアクセス許可を監査する
- 古いアクセス許可を削除する
- 最小特権の原則に基づいてロールを更新する

証明書管理

Elastic Beanstalk 環境に SSL/TLS 証明書に関する以下のプラクティスを実装します。

証明書のライフサイクル

- 証明書管理AWS Certificate Managerに を使用する
- ACM 発行証明書の自動更新を有効にする
- 有効期限通知を設定する

セキュリティ標準

- TLS 1.2 以降を使用する
- HTTPS リスナーAWS のセキュリティポリシーに従う
- 必要に応じて HTTP Strict Transport Security (HSTS) を実装する

セキュリティグループの管理

セキュリティグループのベストプラクティスを実装します。

ルール管理

- すべてのカスタムポート要件を文書化する
- VPC フローログを使用してトラフィックをモニタリングする
- 可能な場合は IP 範囲の代わりにセキュリティグループの参照ルールを使用する

定期的な監査

毎月のレビューを確立して、次のタスクを実行します。

- 未使用のルールを特定して削除する
- 送信元/送信先の要件を検証する

- 重複するルールを確認する

ログ記録とモニタリング

効果的なセキュリティモニタリングを行うには、次のログを設定します。

EC2 インスタンスの Windows イベントログ

```
# Review Security event log
PS C:\migrations_workspace> Get-EventLog -LogName Security -Newest 50

# Check Application event log
PS C:\migrations_workspace> Get-EventLog -LogName Application -Source "IIS*"
```

CloudWatch Logs の統合

CloudWatch Logs エージェントを設定して Windows イベントログを CloudWatch にストリーミングし、一元的なモニタリングとアラートを行います。

永続的な問題については、これらのログを収集し、以下の情報 AWS サポート について にお問い合わせください。

- 環境 ID
- デプロイ ID (該当する場合)
- 関連するエラーメッセージ
- セキュリティ変更のタイムライン

IIS から Elastic Beanstalk への移行マッピングについて

IIS から Elastic Beanstalk への移行には、オンプレミスの Windows サーバー設定を AWS クラウドリソースにマッピングする必要があります。このマッピングを理解することは、移行の成功と移行後の管理に不可欠です。

Elastic Beanstalk の IIS サイトとアプリケーション

IIS では、ウェブサイトはウェブアプリケーションと仮想ディレクトリのコレクションを表し、それぞれに独自の設定とコンテンツがあります。Elastic Beanstalk に移行すると、これらのコンポーネントは次のように変換されます。

IIS ウェブサイト

IIS ウェブサイトは Elastic Beanstalk 内のアプリケーションになります。バインディング、アプリケーションプール、認証設定を含む各ウェブサイトの設定は、Elastic Beanstalk のデプロイマニフェスト () を通じて保持されます`aws-windows-deployment-manifest.json`。

たとえば、Default Web Site や IntranetSite などの複数のサイトがある場合、 は各サイトのコンテンツと設定をeb migrateパッケージ化し、分離を維持します。

コマンドは、アプリケーションへのルーティングリクエストを処理するための適切な Application Load Balancer (ALB) リスナールールを作成します。また、元の IIS バインディングに基づいて適切なポートアクセスを確保するようにセキュリティグループを設定します。

アプリケーションプール

IIS アプリケーションプールは、アプリケーションのワーカープロセスの分離、ランタイム管理、リサイクル機能を提供します。Elastic Beanstalk では、これらは`aws:elasticbeanstalk:environment:process`名前空間で定義され、EC2 インスタンスの IIS を介して設定される環境プロセスにマッピングされます。

移行では、次のような重要なアプリケーションプール設定が保持されます。

- プロセスモデル設定 - ID (ApplicationPoolIdentity、NetworkService、またはカスタムアカウント) 、アイドルタイムアウト設定、プロセスリサイクル間隔
- .NET CLR バージョン設定 - アプリケーションの互換性を確保するために、指定された .NET Framework バージョン (v2.0、v4.0、またはマネージドコードなし) を維持します。
- マネージドパイプラインモード - 統合パイプラインモードまたはクラシックパイプラインモードの設定を保持して、HTTP リクエスト処理アーキテクチャを維持します。
- 詳細設定 - キューの長さ、CPU 制限、ラピッドフェイル保護しきい値、起動時間制限

eb migrate コマンドは、Elastic Beanstalk 環境への移行中にサイトとアプリケーションプール間のマッピングを保持します。

アプリケーションプールがカスタムリサイクルスケジュール (特定の時間またはメモリしきい値) を使用している場合、これらは EC2 インスタンスで適切な IIS 設定を構成するデプロイパッケージの PowerShell スクリプトを通じて実装されます。

ウェブサイトバインディング

クライアントがアプリケーションにアクセスする方法を定義する IIS ウェブサイトバインディングは、次の Application Load Balancer (ALB) 設定に変換されます。

- ポートバインディングは、対応する ALB リスナールールにマッピングされます。
- ホストヘッダー設定は ALB ルーティングルールに変換されます
- SSL 対応サイトは証明書管理に AWS Certificate Manager (ACM) を使用します

仮想ディレクトリとアプリケーションパスの管理

IIS 仮想ディレクトリとアプリケーションは、物理ディレクトリへの URL パスマッピングを提供します。Elastic Beanstalk は、次のコンストラクトを通じてこれらの関係を維持します。

仮想ディレクトリ

移行プロセスでは、仮想ディレクトリの物理パスがデプロイパッケージに保持されます。

パスマッピングは EC2 インスタンスの IIS 設定で設定され、移行後も URL 構造がそのまま維持されます。

システムドライブ以外の物理パス

> **Important**
>
> デフォルトでは、Elastic Beanstalk Windows 環境は C:\ ドライブ (ルートボリューム) のみをプロビジョニングします。現在のバージョンでは、システム以外のドライブ (D:\、E:\ など) にコンテンツがあるアプリケーションは移行をサポートしていません。

eb migrate コマンドは、システム以外のドライブにある物理パスを自動的に検出し、次の例のような潜在的な問題について警告します。

```
ERROR: Detected physical paths on drive D:\ which are not supported in the current
 version:
  - D:\websites\intranet
  - D:\shared\images

Migration of content from non-system drives is not supported. Please relocate this
 content to the C:\ drive before migration. Otherwise, select only those sites that
 are on C:\.
```

アプリケーションがシステム以外のドライブに依存している場合は、移行前にアプリケーションを変更してすべてのコンテンツを C:\ ドライブに保存する必要があります。

ネストされたアプリケーション

ウェブサイトの下にネストされたアプリケーションは、正しいパス設定と適切なアプリケーションプール割り当てでデプロイされます。移行プロセスではすべての `web.config`設定が保持されるため、アプリケーション固有の設定がクラウド環境で期待どおりに機能し続けます。

URL 書き換えとアプリケーションリクエストルーティング (ARR)

IIS デプロイで URL の書き換えまたはアプリケーションリクエストルーティング (ARR) を使用している場合、 は次のルールと設定を通じてこれらの設定eb migrateを処理します。

URL 書き換えルール

`web.config` ファイルからの URL 書き換えルールは、可能な場合は ALB ルーティングルールに変換されます。たとえば、次のエントリは、ホストヘッダーとパスパターンに基づいてトラフィックを指示する ALB リスナールールになります。

```
<!-- Original IIS URL Rewrite Rule -->
<rule name="Redirect to WWW" stopProcessing="true">
    <match url="(.*)" />
    <conditions>
        <add input="{HTTP_HOST}" pattern="^example.com$" />
    </conditions>
    <action type="Redirect" url="http://www.example.com/{R:1}" />
</rule>
```

アプリケーションリクエストのルーティング

ARR 設定は、EC2 インスタンスに ARR 機能をインストールすることで保持されます。移行プロセスは、次のタスクを完了します。

- ソース環境に合わせてプロキシ設定を構成する
- ARR に関連付けられた URL 書き換えルールを維持します

移行アーティファクト構造

を実行するとeb migrate、必要なすべてのデプロイコンポーネントを含む構造化ディレクトリが作成されます。次のリストでは、ディレクトリ構造について説明します。

```
C:\migration_workspace\
### .\migrations\latest\
    ### upload_target\
        ### [SiteName].zip                # One ZIP per IIS site
        ### aws-windows-deployment-manifest.json
        ### ebmigrateScripts\
            ### site_installer.ps1        # Site installation scripts
            ### arr_configuration.ps1     # ARR configuration scripts
            ### permission_handler.ps1    # Permission management
            ### firewall_config.ps1       # Windows Firewall rules
```

`aws-windows-deployment-manifest.json` ファイルは、アプリケーションをデプロイする方法を Elastic Beanstalk に指示するコア設定ファイルです。次の構造例を参照してください。

```
{
    "manifestVersion": 1,
    "deployments": {
        "msDeploy": [
            {
                "name": "Primary Site",
                "parameters": {
                    "appBundle": "DefaultWebSite.zip",
                    "iisPath": "/",
                    "iisWebSite": "Default Web Site"
                }
            }
        ],
        "custom": [
            {
                "name": "ConfigureARR",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\\arr_configuration.ps1"
                    },
                    "uninstall": {
                        "file": "ebmigrateScripts\\noop.ps1"
                    },
                    "restart": {
                        "file": "ebmigrateScripts\\noop.ps1"
                    }
                }
            }
        ]
```

```
    }
}
```

このマニフェストにより、移行の結果が保証されます。

- アプリケーションが正しい IIS パスにデプロイされる
- カスタム設定が適用されます
- サイト固有の設定は保持されます
- デプロイ順序が維持されている

高度な移行シナリオ

このセクションでは、複雑な IIS デプロイの高度な移行シナリオについて説明します。

アプリケーションリクエストルーティング (ARR) を使用したマルチサイト移行

eb migrate コマンドは、移行中に ARR 設定を自動的に検出して保持します。IIS で ARR 設定を識別するとapplicationHost.config、ターゲット EC2 インスタンスに ARR を再インストールして設定するために必要な PowerShell スクリプトが生成されます。

ARR 設定検出

移行プロセスでは、IIS の 3 つの主要な設定セクションを調べます。

- `system.webServer/proxy`: コア ARR プロキシ設定
- `system.webServer/rewrite`: URL 書き換えルール
- `system.webServer/caching`: キャッシュ設定

たとえば、ポート 80 でRouterSite実行されている が へのリクエストをプロキシし、ポート 8081 APIService と 8082 でAdminPortal実行されている一般的な ARR 設定を考えてみましょう。

```
<!-- Original IIS ARR Configuration -->
<rewrite>
    <rules>
        <rule name="Route to API" stopProcessing="true">
            <match url="^api/(.*)$" />
```

```
            <action type="Rewrite" url="http://backend:8081/api/{R:1}" />
        </rule>
        <rule name="Route to Admin" stopProcessing="true">
            <match url="^admin/(.*)$" />
            <action type="Rewrite" url="http://backend:8082/admin/{R:1}" />
        </rule>
    </rules>
</rewrite>
```

次の図は、これらのルールが IIS サーバーのポート 80 の背後に隠れていて、EC2 セキュリティグループを介して公開されていない方法を示しています。Application Load Balancer はポート 80 にのみアクセスでき、そこからのすべてのトラフィックはポート 80 のターゲットグループにルーティングされます。

次のコマンドは、この設定を移行できます。

```
PS C:\migrations_workspace> eb migrate --sites "RouterSite,APIService,AdminPortal" `
    --copy-firewall-config
```

ARR 移行プロセス

移行プロセスでは、いくつかのステップで ARR 設定が保持されます。

設定のエクスポート

このツールは、既存の ARR 設定を 3 つのキー設定セクションから `ebmigrateScripts` ディレクトリに保存されている個別の XML ファイルにエクスポートします。

```
ebmigrateScripts\
### arr_config_proxy.xml
### arr_config_rewrite.xml
### arr_config_caching.xml
```

インストールスクリプト

ARR 設定を処理するために 2 つの PowerShell スクリプトが生成されます。

1. `arr_msi_installer.ps1`: ARR モジュールをダウンロードしてインストールします
2. `arr_configuration_importer_script.ps1`: エクスポートされた ARR 設定をインポートします

デプロイマニフェストの統合

スクリプトは、 のエントリを通じてデプロイプロセスに統合されます`aws-windows-deployment-manifest.json`。

```
{
    "manifestVersion": 1,
    "deployments": {
        "custom": [
            {
                "name": "WindowsProxyFeatureEnabler",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\
\windows_proxy_feature_enabler.ps1"
                    }
                }
            },
            {
                "name": "ArrConfigurationImporterScript",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\
\arr_configuration_importer_script.ps1"
                    }
                }
            }
        ]
    }
}
```

ロードバランサーの統合

移行プロセスは、可能な限り ARR ルールを Application Load Balancer (ALB) リスナールールに変換します。例えば、上記の ARR 設定は、EC2 インスタンスの内部ルーティングを維持しながら、URL パスパターンに基づいてトラフィックをルーティングする ALB ルールになります。

結果として得られる環境は、 AWSのエラスティックインフラストラクチャを活用しながら、ARR ルーティングロジックを維持します。Application Load Balancer が外部トラフィック分散を管理している間、アプリケーションは以前と同じように動作し、ARR は内部ルーティングを処理します。

ホストベースのルーティングを使用した ARR なしのマルチサイト移行

Application Request Routing (ARR) は IIS で複数のサイトを管理するための一般的なアプローチですが、Application Load Balancer のホストベースのルーティング機能を活用することで、ARR なしでマルチサイトデプロイを Elastic Beanstalk に直接移行することもできます。このアプローチは、追加のルーティングレイヤーを排除することで、複雑さを軽減し、パフォーマンスを向上させることができます。

ホストベースのルーティングの概要

このアプローチでは、各 IIS サイトが EC2 インスタンスの外部に公開され、Application Load Balancer はホストヘッダーに基づいてトラフィックを適切なポートに直接ルーティングします。これにより、アプリケーション間の分離を維持しながら ARR が不要になります。

3 つのサイトを持つマルチサイト IIS 設定を考えてみましょう。それぞれに独自のホスト名バインディングがあります。

```
<sites>
    <site name="Default Web Site" id="1">
        <bindings>
            <binding protocol="http" bindingInformation="*:8081:www.example.com" />
        </bindings>
    </site>
    <site name="InternalAPI" id="2">
        <bindings>
            <binding protocol="http" bindingInformation="*:8082:api.internal" />
        </bindings>
    </site>
    <site name="ReportingPortal" id="3">
        <bindings>
            <binding protocol="http" bindingInformation="*:8083:reports.internal" />
        </bindings>
    </site>
</sites>
```

これらのサイトは、EC2 セキュリティグループを介してポート 8081、8082、および 8083 で公開されます。Application Load Balancer は、Load Balancerリスナールール設定に基づいてルーティングされます。

移行プロセス

ARR を使用せずにこの設定を Elastic Beanstalk に移行するには、次の例の eb migrate コマンドを使用します。

```
PS C:\migrations_workspace> eb migrate --sites "Default Web
 Site,InternalAPI,ReportingPortal"
```

移行プロセスでは、ホストヘッダーに基づいてトラフィックを適切なターゲットグループに転送するホストベースのルーティングルールを使用して Application Load Balancer が自動的に設定されます。各ターゲットグループは、EC2 インスタンスの対応するポートにトラフィックを転送します。

1. ホストヘッダー www.example.com → ポート 8081 のターゲットグループ
2. ホストヘッダー api.internal → ポート 8082 のターゲットグループ
3. ポート 8083 のホストヘッダー reports.internal → Target Group

SSL/TLS 設定

SSL/TLS を使用してアプリケーションを保護するには、次の手順を実行します。

1. AWS Certificate Manager(ACM) を使用してドメインの証明書をリクエストします。
2. これらの証明書を使用して、Application Load Balancer で HTTPS リスナーを設定します。
3. 環境設定を更新して、次の設定オプション設定で HTTPS リスナーを含めます。

```
option_settings:
  aws:elb:listener:443:
    ListenerProtocol: HTTPS
    SSLCertificateId: arn:aws:acm:region:account-id:certificate/certificate-id
    InstancePort: 80
    InstanceProtocol: HTTP
```

この設定では、SSL 終了はロードバランサーで発生し、トラフィックは HTTP 経由でインスタンスに転送されます。これにより、クライアントとの安全な接続を維持しながら、証明書管理が簡素化されます。

ベストプラクティス

セキュリティグループ

Application Load Balancer セキュリティグループから IIS サイト (この例では 8081、8082、8083) が使用するポートでのみインバウンドトラフィックを許可するようにセキュリティグループを設定します。

ヘルスチェック

各ターゲットグループのヘルスチェックを設定して、トラフィックが正常なインスタンスにのみルーティングされるようにします。まだ存在しない場合は、アプリケーションごとにヘルスチェックエンドポイントを作成します。

モニタリング

CloudWatch アラームを設定して、各ターゲットグループのヘルスとパフォーマンスを個別にモニタリングします。これにより、個々のアプリケーションに固有の問題を特定できます。

スケーリング

自動スケーリングポリシーを設定するときは、すべてのアプリケーションのリソース要件を考慮してください。1 つのアプリケーションでリソースのニーズが大幅に異なる場合は、別の環境への移行を検討してください。

仮想ディレクトリ管理

eb migrate このコマンドは、IIS アプリケーションを Elastic Beanstalk に移行する間、仮想ディレクトリ構造を保持します。

デフォルトのアクセス許可設定

仮想ディレクトリを移行する場合、 は、以下へのアクセスを ReadAndExecute に付与することで、アクセス許可のベースラインセットeb migrateを確立します。

- IIS_IUSRS
- IUSR
- 認証されたユーザー

たとえば、一般的な仮想ディレクトリ構造を考えてみましょう。

```
<site name="CorporatePortal">
    <application path="/" applicationPool="CorporatePortalPool">
        <virtualDirectory path="/" physicalPath="C:\sites\portal" />
        <virtualDirectory path="/shared" physicalPath="C:\shared\content" />
        <virtualDirectory path="/reports" physicalPath="D:\reports" />
    </application>
</site>
```

パスワードで保護された仮想ディレクトリ

がパスワードで保護された仮想ディレクトリeb migrateを検出すると、警告が発行され、手動による介入が必要になります。

次の設定例では、この例に続く警告レスポンスが発生します。

```
<virtualDirectory path="/secure"
                 physicalPath="C:\secure\content"
                 userName="DOMAIN\User"
                 password="[encrypted]" />
```

```
[WARNING] CorporatePortal/secure is hosted at C:\secure\content which is password-
protected and won't be copied.
```

パスワード保護を維持するには、次のようなカスタムデプロイスクリプトを作成します。

```
# PS C:\migrations_workspace> cat secure_vdir_config.ps1

$vdirPath = "C:\secure\content"
$siteName = "CorporatePortal"
$vdirName = "secure"
$username = "DOMAIN\User"
$password = "SecurePassword"

# Ensure directory exists
if (-not (Test-Path $vdirPath)) {
    Write-Host "Creating directory: $vdirPath"
    New-Item -Path $vdirPath -ItemType Directory -Force
}

# Configure virtual directory with credentials
Write-Host "Configuring protected virtual directory: $vdirName"
```

```
New-WebVirtualDirectory -Site $siteName -Name $vdirName `
    -PhysicalPath $vdirPath -UserName $username -Password $password

# Set additional permissions as needed
$acl = Get-Acl $vdirPath
$rule = New-Object System.Security.AccessControl.FileSystemAccessRule(
    $username, "ReadAndExecute", "ContainerInherit,ObjectInherit", "None", "Allow"
)
$acl.AddAccessRule($rule)
Set-Acl $vdirPath $acl
```

このスクリプトをマニフェストに含めることで、デプロイに追加します。

```
{
    "manifestVersion": 1,
    "deployments": {
        "custom": [
            {
                "name": "SecureVirtualDirectory",
                "scripts": {
                    "install": {
                        "file": "secure_vdir_config.ps1"
                    }
                }
            }
        ]
    }
}
```

カスタムアクセス許可管理

eb migrate コマンドは、デフォルト以外のアクセス許可を必要とするアプリケーションに対応するためのカスタムアクセス許可スクリプトのフレームワークを提供します。

```
$paths = @(
    "C:\sites\portal\uploads",
    "C:\shared\content"
)

foreach ($path in $paths) {
    if (-not (Test-Path $path)) {
```

```
        Write-Host "Creating directory: $path"
        New-Item -Path $path -ItemType Directory -Force
    }

    $acl = Get-Acl $path

    # Add custom permissions
    $customRules = @(
        # Application Pool Identity - Full Control
        [System.Security.AccessControl.FileSystemAccessRule]::new(
            "IIS AppPool\CorporatePortalPool",
            "FullControl",
            "ContainerInherit,ObjectInherit",
            "None",
            "Allow"
        ),
        # Custom Service Account
        [System.Security.AccessControl.FileSystemAccessRule]::new(
            "NT SERVICE\CustomService",
            "Modify",
            "ContainerInherit,ObjectInherit",
            "None",
            "Allow"
        )
    )

    foreach ($rule in $customRules) {
        $acl.AddAccessRule($rule)
    }

    Set-Acl $path $acl
    Write-Host "Custom permissions applied to: $path"
}
```

ベストプラクティス

これらのベストプラクティスに従って、移行を計画、実行、モニタリング、検証します。

移行前計画

移行前に、既存のアクセス許可と認証要件を文書化します。本番環境にデプロイする前に、開発環境でカスタムアクセス許可スクリプトをテストします。

共有コンテンツ管理

共有コンテンツディレクトリの場合、必要なすべてのファイルシステムのアクセス許可がカスタムスクリプトを使用して適切に設定されていることを確認します。共有ストレージ要件には、Amazon FSx for Windows File Server の使用を検討してください。

モニタリングと検証

移行後にアプリケーションログをモニタリングして、仮想ディレクトリへの適切なアクセスを確認します。以下の点に特に注意してください。

- アプリケーションプール ID アクセス
- カスタムサービスアカウントのアクセス許可
- ネットワーク共有接続
- 認証の失敗

カスタムアプリケーションプール設定

eb migrate コマンドは、デフォルトではカスタムアプリケーションプール設定をコピーしません。カスタムアプリケーションプール設定を保持するには、以下の手順でカスタムマニフェストセクションを作成して適用します。

1. 移行アーティファクトのアーカイブを作成します。

```
PS C:\migrations_workspace> eb migrate --archive
```

2. カスタム PowerShell スクリプトを作成して、アプリケーションプールを設定します。

```
# PS C:\migrations_workspace> cat .\migrations\latest\upload_target
\customize_application_pool_config.ps1

$configPath = "$env:windir\System32\inetsrv\config\applicationHost.config"

[xml]$config = Get-Content -Path $configPath

$newPoolXml = @"
<!-- Original IIS Configuration -->
<applicationPools>
    <add name="CustomPool"
         managedRuntimeVersion="v4.0"
         managedPipelineMode="Integrated">
```

```
        <processModel identityType="SpecificUser"
                     userName="AppPoolUser"
                     password="[encrypted]" />
        <recycling>
            <periodicRestart time="00:00:00">
                <schedule>
                    <add value="02:00:00" />
                    <add value="14:00:00" />
                </schedule>
            </periodicRestart>
        </recycling>
    </add>
</applicationPools>
"@
$newPoolXmlNode = [xml]$newPoolXml

# Find the applicationPools section
$applicationPools = $config.SelectSingleNode("//configuration/
system.applicationHost/applicationPools")

# Import the new node into the document
$importedNode = $config.ImportNode($newPoolXmlNode.DocumentElement, $true)
$applicationPools.AppendChild($importedNode)

# Save the changes
$config.Save($configPath)

Write-Host "ApplicationHost.config has been updated successfully."
```

3. `aws-windows-deployment-manifest.json` ファイルを更新して、カスタムスクリプトを含めます。

```
{
    "manifestVersion": 1,
    "deployments": {
        ...
        "custom": [
            ...,
            {
                "name": "ModifyApplicationPoolConfig",
                "description": "Modify application pool configuration from source
 machine to remove",
                "scripts": {
```

```
                    "install": {
                        "file": "customize_application_pool_config.ps1"
                    },
                    "restart": {
                        "file": "ebmigrateScripts\\noop.ps1"
                    },
                    "uninstall": {
                        "file": "ebmigrateScripts\\noop.ps1"
                    }
                }
            }
        ]
    }
}
```

4. 更新されたアーカイブディレクトリを使用して環境を作成します。

```
PS C:\migrations_workspace> eb migrate `
    --archive-dir '.\migrations\latest\upload_target\'
```

`--archive-dir` 引数は、新しいアーカイブの作成を回避するために、以前に作成したソースコードを使用するeb migrateように に指示します。

以前のバージョンのデプロイ

は、Elastic Beanstalk のタイムスタンプ付きディレクトリとアプリケーションバージョンを通じて移行の履歴eb migrateを維持します。各移行では、必要に応じてデプロイできる一意の zip ファイルが作成されます。

```
PS C:\migrations_workspace> ls .\migrations\

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d----l        3/18/2025  10:34 PM                latest
d-----        3/16/2025   5:47 AM                migration_1742104049.479849
d-----        3/17/2025   9:18 PM                migration_1742246303.18056
d-----        3/17/2025   9:22 PM                migration_1742246546.565739
...
d-----        3/18/2025  10:34 PM                migration_1742337258.30742
```

latest シンボリックリンクは常に、最後に作成された移行アーティファクトディレクトリを指します。関連するアプリケーションログとエラーログに加えて、各移行アーティファクトディレクトリには、Elastic Beanstalk にデプロイできるupload_target.zipファイルも含まれています。

```
PS C:\migrations_workspace> ls .\migrations\latest\

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        3/18/2025  10:34 PM                upload_target
-a----        3/18/2025  10:34 PM          13137 application.log
-a----        3/18/2025  10:34 PM              0 error.log
-a----        3/18/2025  10:34 PM        1650642 upload_target.zip
```

を使用してupload_target.zipファイルをデプロイできますeb migrate。

```
PS C:\migrations_workspace> eb migrate --zip .\migrations\latest\upload_target.zip
```

トラブルシューティングと診断

このセクションでは、IIS アプリケーションの Elastic Beanstalk への移行中に発生する可能性がある一般的な問題のトラブルシューティングに関するガイダンスを提供します。

EC2 キーペアを環境に関連付ける

Amazon EC2 キーペアを使用して、Elastic Beanstalk アプリケーション用にプロビジョニングされた Amazon Elastic Compute Cloud (Amazon EC2) インスタンスに安全にログインできます。キーペアを作成する手順については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 を使用したキーペアの作成」を参照してください。

キー名を に指定すると、Elastic Beanstalk 環境をキーペアに関連付ける効果eb migrateがあります。セキュリティ上の理由から、EC2 インスタンスのセキュリティグループでポート 3389 は開かれません。最初の移行eb config後に、ポート 3389 から までのトラフィックを許可する追加の EC2 セキュリティグループを関連付けることができます。

```
PS C:\migrations_workspace> eb migrate  `
    --keyname "my-keypair"  `
    --verbose
```

キーペアを作成すると、Amazon EC2 はパブリックキーのコピーを保存します。環境インスタンスに接続するためにこれを使用する必要がなくなった場合は、Amazon EC2 から削除できます。詳細については、「Amazon EC2 ユーザーガイド」の「キーペアの削除」を参照してください。

Windows Amazon EC2 インスタンスへの接続の詳細については、「Windows インスタンスへの接続」を参照してください。

ログへのアクセス

EB CLI には、EC2 インスタンスにログインせずに Elastic Beanstalk 環境からログを取得するために使用できる eb logs機能が用意されています。の実行後eb migrate、 eb logs --zip コマンドを発行して、ログをダウンロードして `.elasticbeanstalk\logs` ディレクトリに保存できます。

または、Elastic Beanstalk AWS コンソールからログを表示することもできます。詳細については、「Elastic Beanstalk 環境の Amazon EC2 インスタンスからのログの表示」を参照してください。

クライアント側のアーティファクトへのアクセス

eb migrate コマンドは、 によって生成されたアプリケーションログとエラーログを移行アーティファクトディレクトリmsdeploy内に保存します。

```
./migrations/
### latest -> migration_20240308_123456/
### migration_20240308_123456/
    ### application.log
    ### error.log
    ### upload_target\
```

環境の状態のモニタリング

Elastic Beanstalk は、拡張ヘルスモニタリング機能を使用してヘルスをモニタリングするのに役立ちます。これは、CPU 使用率、レイテンシー、リクエスト数、レスポンスコードなどの組み込みメトリクスを活用して、アプリケーションインスタンスの運用ステータスを継続的に追跡する自動ヘルスモニタリングシステムです。

ヘルスモニタリングシステムは、エージェントベースのアプローチを使用してインスタンスレベルのデータを収集し、リアルタイムのログ記録とアラートと統合します。Elastic Load Balancing (ELB) と Auto Scaling は、ヘルスステータスの変化に動的に対応し、高可用性と耐障害性を確保します。拡張ヘルスレポートなどの高度なモニタリングモードは、アプリケーションの動作をきめ細かく可視化し、プロアクティブなトラブルシューティングと自動復旧メカニズムを可能にします。

EB CLI eb health コマンドを実行して、環境の状態を表示します。以下の情報が表示されます。

- インスタンスのヘルスステータス
- アプリケーションレスポンスメトリクス
- システムリソースの使用率
- 最近のデプロイイベント

EC2 パフォーマンスの最適化

デフォルトでは、 は c5.2xlarge インスタンスタイプeb migrateを選択して、Elastic Beanstalk を初めて使用するときに最適なエクスペリエンスを提供します。この動作は --instance-type引数で上書きできます。

```
PS C:\migrations_workspace> eb migrate `
    --instance-type "t3.large"
```

本番環境では、インスタンスタイプを選択するときに次の要素を考慮してください。

- アプリケーションのメモリ要件
- ワークロードを処理するための CPU 要件
- ネットワークパフォーマンスのニーズ
- コスト最適化の目標

EBS ボリューム設定

デフォルトでは、Elastic Beanstalk は環境のルートブロックデバイスボリューム (`C:\`) のみを作成します。--ebs-snapshots オプションを使用して、追加の Amazon Elastic Block Store スナップショットボリュームを渡すことができます。

```
PS C:\migrations_workspace> eb migrate `
    --ebs-snapshots "snap-123456789abc"
```

Elastic Beanstalk でブロックデバイスマッピングを設定する方法の例については、ブログ記事「Elastic Beanstalk 環境でエフェメラルボリュームと EBS ボリュームをカスタマイズする」を参照してください。

ストレージ要件が高いアプリケーションの場合は、次のオプションを検討してください。

- 永続データに EBS ボリュームを使用する
- 静的コンテンツ用の Amazon S3 の実装
- 共有ファイルシステムに Amazon FSx for Windows File Server を使用する

一般的な の問題と解決策

イベント： Web Deploy のインストールがありません

Web Deploy に関連するエラーが見つからない場合は、Microsoft Web Platform Installer から Web Deploy 3.6 以降をインストールします。次の の例では、考えられるエラーメッセージを表示します。

```
Couldn't find msdeploy.exe. Follow instructions here: https://learn.microsoft.com/en-us/iis/install/installing-publishing-technologies/installing-and-configuring-web-deploy
```

イベント： 移行中のアクセス許可の問題

アクセス許可関連のエラーが発生した場合は、EB CLI を管理権限で実行していることを確認してください。次の の例では、考えられるエラーメッセージを表示します。

```
[ERROR] Access to the path 'C:\inetpub\wwwroot\web.config' is denied.
```

イベント： アプリケーションプール ID の問題

アプリケーションプール ID の問題が原因でアプリケーションを起動できない場合は、「」に示すように、アプリケーションプール ID を設定するカスタムスクリプトを作成しますthe section called “カスタムアプリケーションプール設定”。

イベント： SSL 証明書設定エラー

HTTPS バインディングが機能しない場合は、 eb mibrateオプション`--ssl-certificates`パラメータを使用して有効な ACM 証明書 ARN を指定していることを確認してください。

イベント： 環境作成タイムアウト

環境の作成がタイムアウトした場合は、 AWS マネジメントコンソールの AWS CloudFormation イベントで特定のリソース作成の失敗を確認します。一般的な原因には、VPC 設定の問題やサービスの制限などがあります。

サポート情報

解決できない問題が発生した場合は、 に連絡する前に次の情報を収集 AWS サポート してください。

- 環境 ID (`eb status`)
- アプリケーションログ (`eb logs --zip`)
- からの移行アーティファクト `.\migrations\latest\`
- ソース IIS 設定 (の出力`eb migrate explore --verbose`)
- 詳細なエラーメッセージ

Elastic Beanstalk のトラブルシューティングの詳細については、「」を参照してください[Elastic Beanstalk 環境のトラブルシューティング](#)。

移行オプションの比較: EB CLI と AWS Application Migration Service

AWS は、Windows アプリケーションをクラウドに移行するための複数のパスを提供します。このセクションでは、EB CLI の eb migrate コマンドと AWS Application Migration Service (MGN) の 2 つの主要なオプションを比較します。これらのアプローチの違いを理解することは、特定のニーズに最適な移行戦略を選択するのに役立ちます。

移行オプションの比較

機能	EB CLI (eb migrate)	AWS Application Migration Service (MGN)
主な焦点	IIS ウェブサイトとアプリケーションのアプリケーションレベルの移行	マシン全体のサーバーレベルのリホスト (物理サーバー、仮想サーバー、クラウドサーバー)
に最適	最小限の再設定で Elastic Beanstalk に直接移行する IIS アプリケーション	多数のサーバーや複雑なインフラストラクチャを含む大規模な移行

機能	EB CLI (eb migrate)	AWS Application Migration Service (MGN)
検出アプローチ	IIS サイト、アプリケーション、および設定のアプリケーションレベルの検出	オペレーティングシステムやアプリケーションを含むマシン全体のサーバーレベルのレプリケーション
ターゲット環境	Windows アプリケーション用に最適化された Elastic Beanstalk 環境を直接作成および設定する	Elastic Beanstalk を操作するために追加の設定が必要な EC2 インスタンスを作成します
設定の保存	IIS 固有の設定 (サイト、アプリケーションプール、バインディング) を自動的に保持します	不要なコンポーネントを含むサーバー設定全体を保持します
デプロイモデル	Elastic Beanstalk のベストプラクティスを使用してデプロイされたアプリケーションを使用して、クリーンな Elastic Beanstalk 環境を作成します。	クラウドオペレーションの最適化が必要なソースサーバーのレプリカを作成します。
移行のスケール	特定のアプリケーションのターゲット移行に最適	多数のサーバーの大規模な移行用に設計
移行ステップ	最小。環境は Elastic Beanstalk 管理ツールで使用可能	SSM 起動後アクションの実行など、Elastic Beanstalk と統合するには追加のステップが必要です

各移行オプションを使用するタイミング

次の要件eb migrateがある場合は、 を選択します。

- サーバー全体ではなく特定の IIS アプリケーションを移行したい
- 目標は、Elastic Beanstalk をアプリケーション管理プラットフォームとして採用することです。
- 簡単なスケーリング、デプロイ、モニタリングなど、Elastic Beanstalk のマネージドプラットフォーム機能を活用する場合

- クラウドネイティブオペレーションの AWS ベストプラクティスに従うクリーンなデプロイが望ましい
- 移行後の設定作業を最小限に抑えたい

次の要件 AWS Application Migration Service がある場合は、 を選択します。

- 多数のサーバーを移行する必要がある
- 正確に保持する必要がある複雑なサーバー設定がある
- アプリケーションに互換性の問題があり、サーバー環境を正確に維持する必要がある
- アプリケーションへの最小限の変更で「リフトアンドシフト」したい
- 移行後にアプリケーションのリファクタリングまたは最適化を計画している

移行ワークフローの比較

EB CLI (eb migrate) ワークフロー：

1. EB CLI をソース IIS サーバーまたは踏み台ホストにインストールします。
2. eb migrate を実行して IIS アプリケーションを検出します。
3. コマンドは、アプリケーションと設定をパッケージ化します。
4. Elastic Beanstalk 環境は、適切なリソースを使用して作成されます。
5. アプリケーションは新しい環境にデプロイされます。
6. Elastic Beanstalk ツールを使用して、アプリケーションをすぐに管理できます。

AWS Application Migration Service ワークフロー：

1. AWS レプリケーションエージェントをソースサーバーにインストールします。
2. データレプリケーションを設定してテストします。
3. テストインスタンスを起動して機能を確認します。
4. カットオーバーをスケジュールします AWS。
5. 本番稼働用インスタンスを起動します。
6. 起動後のアクションを実行してクラウドを最適化します。
7. Elastic Beanstalk がターゲットプラットフォームである場合、Elastic Beanstalk と統合するには追加の設定が必要です。

結論

Elastic Beanstalk は、 上の Windows プラットフォームアプリケーションに推奨される送信先であり AWS、デプロイ、スケーリング、管理を簡素化するマネージド環境を提供します。eb migrate コマンドは、アプリケーション設定を保持する自動検出と設定を使用して、IIS アプリケーション用の Elastic Beanstalk への直接パスを提供します。

AWS Application Migration Service は大規模なサーバー移行に強力な機能を提供しますが、Elastic Beanstalk と統合するには追加のステップが必要です。Elastic Beanstalk がターゲットプラットフォームであるほとんどの IIS アプリケーション移行では、 は Elastic Beanstalk のマネージドサービスモデルに沿ったより合理化されたアプローチeb migrateを提供します。

スケール、複雑さ、希望するエンドステートアーキテクチャなどの要因を考慮して、特定の要件に最も適した移行アプローチを選択します AWS。

詳細については AWS Application Migration Service、「 AWS Application Migration Service ユーザーガイド」の「What is AWS Application Migration Service？」を参照してください。

Elastic Beanstalk 環境のトラブルシューティング

この章では、Elastic Beanstalk 環境の問題をトラブルシューティングするためのガイダンスを提供します。以下の情報を提供します。

- AWS Systems Manager ツールの概要と、トラブルシューティングのステップと推奨事項を出力する事前定義された Elastic Beanstalk ランブックを実行する手順。
- 環境ステータスが低下した場合に実行できるアクションと確認できるリソースに関する一般的なガイダンス。
- 主題カテゴリごとのより具体的なトラブルシューティングのヒント。

> **Note**
>
> 環境の状態が赤に変わった場合は、まず、定義済みのランブックを含む AWS Systems Manager ツールを使用して Elastic Beanstalk のトラブルシューティングを行うことをお勧めします。詳細については、「Systems Manager ツールの使用」を参照してください。

トピック

- AWS Systems Manager Elastic Beanstalk ランブックの使用
- Elastic Beanstalk 環境のトラブルシューティングに関する一般的なガイダンス
- 環境変数を使用してシークレットとパラメータにアクセスする環境
- 環境の作成とインスタンスの起動
- デプロイ
- 健康
- 設定
- Docker コンテナのトラブルシューティング
- よくある質問
- 共通エラー
- デプロイエラー

AWS Systems Manager Elastic Beanstalk ランブックの使用

Systems Manager を使用して Elastic Beanstalk 環境のトラブルシューティングを行うことができます。Systems Manager は、お客様にすぐに使用を開始していただけるように、Elastic Beanstalk 用に事前定義されたオートメーションランブックを用意しています。オートメーションランブックは、環境のインスタンスやその他の AWS リソースで実行するアクションを定義する Systems Manager ドキュメントの一種です。

このドキュメント `AWSSupport-TroubleshootElasticBeanstalk` は、Elastic Beanstalk 環境のパフォーマンスが低下する可能性のある一般的な問題をいくつか特定するのに役立つように設計されたオートメーションランブックです。そうするためには、EC2 インスタンス、VPC、 AWS CloudFormation スタック、ロードバランサー、Auto Scaling グループ、セキュリティグループルール、ルートテーブル、ACL に関連するネットワーク設定など、環境のコンポーネントをチェックします。

また、バンドルされたログファイルを環境から AWS サポートにアップロードするオプションもあります。

詳細についてはAWS Systems Manager「 自動 ランブックリファレンス」の`AWSSupport-TroubleshootElasticBeanstalk`を参照してください。

Systems Manager を使用して **`AWSSupport-TroubleshootElasticBeanstalk`** ランブックを実行する

> **Note**
>
> この手順は、Elastic Beanstalk 環境 AWS リージョン があるのと同じ で実行します。

1. AWS Systems Manager コンソールを開きます。
2. ナビゲーションペインの [変更管理] で、[オートメーション] を選択します。
3. [Execute automation (自動化の実行)] を選択してください。
4. [Amazon が所有] タブの[オートメーションドキュメント] 検索ボックスに、「`AWSSupport-TroubleshootElasticBeanstalk`」を入力します。
5. [AWSSupport-TroubleshootElasticBeanstalk] カードを選択してから、[次へ] を選択します。
6. [実行] を選択します。
7. [入力パラメータ] セクションで、以下の操作を行います。

a. [AutomationAssumeRole] ドロップダウンから、System Manager がユーザーに代わってアクションを実行できるようにするロールの ARN を選択します。
b. [ApplicationName] に、Elastic Beanstalk アプリケーションの名前を入力します。
c. [環境名] には、Elastic Beanstalk 環境を入力します。
d. （オプション) S3UploaderLink の場合、 AWS サポートエンジニアからログ収集用の S3 リンクが提供されている場合は、リンクを入力します。

8. [実行] を選択してください。

   いずれかのステップが失敗した場合は、失敗したステップの [ステップ ID] 列の下にあるリンクを選択します。これにより、ステップの [実行詳細] ページが表示されます。[VerificationErrorMessage] セクションには、注意が必要な手順の概要が表示されます。たとえば、`IAMPermissionCheck` には警告メッセージが表示される場合があります。この場合、[AutomationAssumeRole] ドロップダウンで選択したロールに必要な権限があるかどうかを確認できます。

すべての手順が正常に完了すると、出力にはトラブルシューティングの手順および環境を正常な状態に復元するための推奨事項が表示されます。

Elastic Beanstalk 環境のトラブルシューティングに関する一般的なガイダンス

エラー メッセージは、コンソールの [イベント] ページ、ログ、または [ヘルス] ページに表示されることがあります。また、最近の変更によってパフォーマンスが低下した環境から回復するための対策を講じることもできます。対象環境のヘルスステータスが「赤色」に変わった場合は、以下の操作を試してください。

- 環境の オペレーションがテキスト を含むエラーを返す場合は`The stack stack_id associated with environment environment-ID is in stack-status state`、トラブルシューティングのヘルプElastic Beanstalk 環境を無効な状態から復旧するについて「」を参照してください。
- 環境の オペレーションがテキスト を含むエラーを返した場合は`Environment environment-name associated CloudFormation stack stack_arn does not exist`、環境を終了して別のオペレーションを作成します。
- 最新の環境イベントを確認します。デプロイ、負荷、設定の問題に関する Elastic Beanstalk からのメッセージは、多くの場合、ここに表示されています。

- 最近の、環境に関する変更履歴を確認します。変更履歴には、環境に加えられたすべての設定変更の内容がリストされています。さらに、変更を行った IAM ユーザーや、設定が行われたパラメータなど、その他の情報も含まれています。
- ログをプルして、最新のログファイルエントリを表示します。ウェブサーバーのログには、受信リクエストおよびエラーに関する情報が含まれています。
- インスタンスに接続し、システムリソースをチェックします。
- アプリケーションの以前の稼働バージョンにロールバックします。
- 最新の設定変更を元に戻すか、保存した設定を復元します。
- 新しい環境をデプロイします。この新しい環境に特に問題がないと確認できたら、CNAME スワップを実行してその環境にトラフィックをルーティングし、以前の環境のデバッグを続けます。

環境変数を使用してシークレットとパラメータにアクセスする環境

イベント： インスタンスのデプロイで 1 つ以上のシークレットを取得できませんでした

このメッセージは、Elastic Beanstalk がアプリケーションのデプロイ中に指定された 1 つ以上のシークレットを取得できなかったことを示します。

- 環境変数設定の ARN 値で指定されたリソースが存在することを確認します。
- Elastic Beanstalk EC2 インスタンスプロファイルロールに、リソースにアクセスするために必要な IAM アクセス許可があることを確認します。
- このイベントが `RestartAppServer`オペレーションによってトリガーされた場合、問題が修正されたら、`RestartAppServer`呼び出しを再試行して問題を解決します。
- イベントが `UpdateEnvironment`呼び出しによってトリガーされた場合は、`UpdateEnvironment`オペレーションを再試行します。

これらのコマンドの例については、AWS CLI 「Elastic Beanstalk の例」を参照してください。これらのオペレーションの API アクションの詳細については、AWS Elastic Beanstalk 「 API リファレンス」を参照してください。

イベント： インスタンスのデプロイで、このプラットフォームではサポートされていない 1 つ以上の複数行の環境値が検出されました

複数行変数は、Docker および ECS マネージド Docker プラットフォームを除き、Amazon Linux 2 プラットフォームではサポートされていません。続行できるオプションについては、「」を参照してください複数行の値。

イベント： シークレットが指定されると CreateEnvironment が失敗する

がCreateEnvironment失敗し、環境変数としてシークレットがある場合は、根本的な問題に対処してから、 UpdateEnvironment を使用して環境設定を完了する必要があります。この状況では環境を起動するには十分ではないためRestartAppServer、 を使用しないでください。これらのコマンドの例については、AWS CLI 「Elastic Beanstalk の例」を参照してください。これらのオペレーションの API アクションの詳細については、AWS Elastic Beanstalk 「 API リファレンス」を参照してください。

環境の作成とインスタンスの起動

イベント: 環境を起動できませんでした

このイベントは、Elastic Beanstalk が環境を起動しようとし、その途中でエラーが起こったときに発生します。[イベント] ページの以前のイベントが、この問題の根本的な原因について警告します。

イベント: 環境の作成オペレーションは完了しましたが、コマンドがタイムアウトしました。タイムアウト期間を長くしてみてください。

インスタンス上でコマンドを実行する、大きなファイルをダウンロードする、あるいはパッケージをインストールするために設定ファイルを使用すると、アプリケーションのデプロイに時間がかかる場合があります。コマンドタイムアウトを増やして、デプロイ中にアプリケーションが実行開始できる時間を多くしてください。

イベント: 以下のリソースで [AWSEBInstanceLaunchWaitCondition] を作成できませんでした

このメッセージは、環境の Amazon EC2 インスタンスが、正常に起動した Elastic Beanstalk と通信しなかったことを示します。これは、インスタンスにインターネットの接続がない場合に発生します。プライベート VPC サブネットでインスタンスを起動するように環境を設定した場合は、インスタンスに対して Elastic Beanstalk への接続を許可する NAT がサブネットにあることを確認します。

イベント: このリージョンではサービスロールが必要です。環境にサービスロールオプションを追加してください。

Elastic Beanstalk はサービスロールを使用して環境のリソースをモニタリングし、マネージド型プラットフォーム更新をサポートします。詳細については「Elastic Beanstalk サービスロールの管理」を参照してください。

デプロイ

問題: デプロイ中にアプリケーションが使用不可になります

Elastic Beanstalk はドロップインアップグレードプロセスを使用するため、数秒間のダウンタイムが生じることがあります。ローリングデプロイを使用して、運用環境におけるデプロイの影響を最小化します。

イベント： Elastic Beanstalk AWS アプリケーションバージョンの作成に失敗しました

アプリケーションソースバンドルが大きすぎるか、アプリケーションバージョンクォータに達した可能性があります。

イベント: 環境更新オペレーションは完了しましたが、コマンドがタイムアウトしました。タイムアウト期間を長くしてみてください。

インスタンス上でコマンドを実行する、大きなファイルをダウンロードする、あるいはパッケージをインストールするために設定ファイルを使用すると、アプリケーションのデプロイに時間がかかる場合があります。コマンドタイムアウトを増やして、デプロイ中にアプリケーションが実行開始できる時間を多くしてください。

健康

イベント: CPU 使用率が 95.00% を超える

実行するインスタンスを増やすか、別のインスタンスタイプを選択してください。

イベント: Elastic Load Balancer awseb-*myapp* に正常なインスタンスがない

アプリケーションが機能しているようであれば、アプリケーションのヘルスチェック URL が正しく設定されていることを確認します。そうでなければ、[Health] 画面および環境ログで詳細を確認します。

イベント: Elastic Load Balancer awseb-*myapp* が見つかりません

環境のロードバランサーが帯域外に削除された可能性があります。環境のリソースに対する変更は、Elastic Beanstalk によって提供される設定オプションおよび拡張可能性のみにしてください。環境を再構築するか、新しいインスタンスを起動してください。

イベント: EC2 インスタンス起動エラー。新しい EC2 インスタンスの起動の待機中...

環境のインスタンスタイプの可用性が低いか、アカウントのインスタンスクォータに達した可能性があります。サービスヘルスダッシュボードを調べ、Elastic Compute Cloud (Amazon EC2) サービスが緑色になっていることを確認してください。なっていない場合はクォータの引き上げをリクエストしてください。

設定

イベント： 環境*stack_id*に関連付けられたスタック*environment-ID*が *stack-status*状態です

環境の基盤となる AWS CloudFormation スタックは *_FAILED ステータスの場合があります。環境で Elastic Beanstalk オペレーションを続行するには、このステータスを修正する必要があります。詳細については、「Elastic Beanstalk 環境を無効な状態から復旧する」を参照してください。

イベント: Elastic Load Balancing Target オプションおよび Application Healthcheck URL オプションの値を使用して、Elastic Beanstalk 環境を設定することはできません。

`Target` 名前空間の `aws:elb:healthcheck` オプションは廃止されました。`Target` オプション名前空間を環境から削除してもう一度更新してください。

イベント： ELB を同じ AZ 内の複数のサブネットにアタッチすることはできません

このメッセージは、同じアベイラビリティーゾーン内のサブネット間でロードバランサーを移動しようとすると表示される場合があります。ロードバランサーのサブネットを変更するには、元のアベイラビリティーゾーンの外に移動してから、必要なサブネットを用意した元のアベイラビリティーゾーンに戻す必要があります。この処理中は、すべてのインスタンスは AZ 間で移行されるため、長時間のダウンタイムが発生します。代わりに、新しい環境を作成して CNAME のスワップを実行することを検討してください。

Docker コンテナのトラブルシューティング

イベント: Docker イメージを取得できませんでした :latest: 無効なリポジトリ名です ()。[a-z0-9-_.] のみが許可されています。詳細については、ログを追跡します。

JSON 検証ツールを使用して `dockerrun.aws.json` ファイルの構文をチェックします。また、「Elastic Beanstalk へのデプロイ用に Docker イメージを準備する」で説明されている要件を参照して dockerfile の内容を確認します。

イベント: Dockerfile に EXPOSE ディレクティブが見つかりません。デプロイを中止します

`Dockerfile` または `dockerrun.aws.json` ファイルでコンテナポートが宣言されていません。`EXPOSE` インストラクション (`Dockerfile`) または `Ports` ブロック (`dockerrun.aws.json` ファイル) を使用して、受信トラフィックに対してポートを開きます。

イベント: *#####*から認証資格情報*#####*をダウンロードできませんでした

`dockerrun.aws.json` ファイルは `.dockercfg` ファイルに無効な EC2 キーペアや S3 バケットを指定します。または、インスタンスプロファイルに S3 バケットの GetObject の権限がありません。`.dockercfg` ファイルに有効な S3 バケットと EC2 キーペアが含まれていることを確認します。インスタンスプロファイル内の IAM ロールに `s3:GetObject` 操作を許可します。詳細については、Elastic Beanstalk インスタンスプロファイルの管理 を参照してください

イベント: 以下の理由により、アクティビティの実行に失敗しました。警告: 認証設定ファイルが無効です

認証ファイル (`config.json`) が正しくフォーマットされていません。「イメージリポジトリを使用した認証」を参照してください。

よくある質問

質問:アプリケーション URL を、myapp.us-west-2.elasticbeanstalk.com から www.myapp.com に変更したい。

DNS サーバーで、**`www.mydomain.com CNAME mydomain.elasticbeanstalk.com`** などの CNAME レコードを登録します。

質問: Elastic Beanstalk アプリケーションに特定のアベイラビリティーゾーンを指定できない。

API、CLI、Eclipse プラグイン、または Visual Studio プラグインを使用すると、特定のアベイラビリティーゾーンを指定できます。Elastic Beanstalk コンソールを使用してアベイラビリティーゾーンを指定する方法については、「Elastic Beanstalk 環境インスタンスの Auto Scaling 」を参照してください。

質問: 環境のインスタンスタイプを変更したい

環境のインスタンスタイプを変更するには、[環境設定] ページに移動し、[インスタンス] 設定カテゴリにある [編集] をクリックします。新しいインスタンスタイプを選択し、[適用] をクリックして環境を更新します。その後、実行中のすべてのインスタンスが Elastic Beanstalk により終了され、新しいインスタンスに置き換えられます。

質問: いずれかのユーザーが環境設定を変更したかどうかを知るにはどうすればよいですか?

この情報を表示するには、Elastic Beanstalk コンソールのナビゲーションペインで [変更履歴] をクリックして、すべての環境の設定変更のリストを表示します。このリストから、変更の日時、設定されたパラメータと変更後の値、その変更を行った IAM ユーザーが確認できます。詳細については、「変更履歴」を参照してください。

質問: インスタンスの終了時に Amazon EBS ボリュームが削除されないようにしたい。

環境内のインスタンスは Amazon EBS を使用して保管されますが、Auto Scaling によってインスタンスが終了されると、ルートボリュームが削除されます。ご自身のインスタンス上に、状態やその他のデータを保存することは推奨されません。必要に応じて、「AWS CLI リファレンス」で説明されている`$ aws ec2 modify-instance-attribute -b '/dev/sdc=<vol-id>:false`ように、AWS CLIでボリュームが削除されないようにできます。

質問: Elastic Beanstalk アプリケーションから個人情報を削除したい。

AWS Elastic Beanstalk アプリケーションが使用する リソースは、個人情報を保存する場合があります。環境を終了すると、Elastic Beanstalk で作成されたリソースが終了されます。設定ファイルを使用して、追加したリソースも終了します。ただし、Elastic Beanstalk 環境の外部で AWS リソースを作成し、アプリケーションに関連付ける場合は、アプリケーションが保存した可能性のある個人情報が保持されないように手動で確認する必要がある場合があります。このデベロッパーガイドを通して追加リソースの作成について取り扱う際は、それらの削除を検討する必要がある場合についても必ず言及しています。

共通エラー

このトピックでは、EB CLI の使用時に表示される一般的なエラーメッセージと考えられる解決策を示しています。ここに示していないエラーメッセージが表示される場合は、[フィードバック] リンクを使用してお知らせください。

エラー: Git コマンドの処理中にエラーが発生しました。エラーコード: 128 エラー: 致命的: 有効なオブジェクト名 HEAD ではありません

原因: Git リポジトリを初期化したが、まだコミットしていない場合にこのエラーメッセージが表示されます。プロジェクトフォルダーに Git リポジトリが含まれていると、EB CLI は HEAD リビジョンを検索します。

解決策: ステージングエリアにプロジェクトフォルダ内のファイルを追加してコミットします。

```
~/my-app$ git add .
~/my-app$ git commit -m "First commit"
```

エラー: このブランチにはデフォルト環境がありません。「eb status my-env-name」と入力して環境を指定するか、「eb use my-env-name」と入力してデフォルト環境を設定する必要があります。

原因: Git で新しいブランチを作成すると、デフォルトでは Elastic Beanstalk 環境にアタッチされません。

解決策: eb list を実行して、利用可能な環境のリストを表示します。その後、eb use ***env-name*** を実行して、利用可能な環境のいずれかを使用します。

エラー: 2.0 以上のプラットフォームにはサービスロールが必要です。サービスロールは、--service-role オプションで指定できます

原因: eb create で環境名 (eb create my-env など) を指定した場合、EB CLI はサービスロールの自動作成を試みません。デフォルトのサービスロールがない場合、上記のエラーが表示されます。

解決策: 環境名を指定しないで eb create を実行し、プロンプトに従ってデフォルトのサービスロールを作成します。

デプロイエラー

Elastic Beanstalk デプロイは、404 (アプリケーションの起動に失敗した場合) または 500 (ランタイムにアプリケーションが失敗した場合) のレスポンスで応答する場合があります。多くの一般的な問題をトラブルシューティングするには、EB CLI を使用して、デプロイのステータスの確認、ログの表示、SSH を使用した EC2 インスタンスへのアクセスの取得、またはアプリケーション環境の AWS マネジメントコンソールページを開くことができます。

EB CLI を使用してデプロイのトラブルシューティングを行うには

1. eb status を実行して、現在のデプロイのステータスおよび EC2 ホストの状態を確認します。例:

```
$ eb status --verbose

Environment details for: python_eb_app
  Application name: python_eb_app
  Region: us-west-2
  Deployed Version: app-150206_035343
  Environment ID: e-wa8u6rrmqy
  Platform: 64bit Amazon Linux 2014.09 v1.1.0 running Python 2.7
  Tier: WebServer-Standard-
  CNAME: python_eb_app.elasticbeanstalk.com
  Updated: 2015-02-06 12:00:08.557000+00:00
  Status: Ready
  Health: Green
```

```
  Running instances: 1
      i-8000528c: InService
```

> **Note**
>
> `--verbose` スイッチを使用すると、実行中のインスタンスのステータスに関する情報が表示されます。このスイッチを指定しない場合、eb status では、環境に関する全般的な情報のみが表示されます。

2. 環境のヘルス情報を表示するには、eb health を実行します。

```
$ eb health --refresh
 elasticBeanstalkExa-env                                    Degraded
 2016-03-28 23:13:20
WebServer
    Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        2        0        2        1        0        0        0

  instance-id   status     cause
    Overall     Degraded  Incorrect application version found on 3 out of 5
 instances. Expected version "Sample Application" (deployment 1).
  i-d581497d    Degraded  Incorrect application version "v2" (deployment 2).
 Expected version "Sample Application" (deployment 1).
  i-d481497c    Degraded  Incorrect application version "v2" (deployment 2).
 Expected version "Sample Application" (deployment 1).
  i-136e00c0    Severe    Instance ELB health has not been available for 5 minutes.
  i-126e00c1    Ok
  i-8b2cf575    Ok

  instance-id   r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75
 p50     p10
    Overall     646.7   100.0    0.0    0.0    0.0    0.003    0.002    0.001
 0.001   0.000
  i-dac3f859    167.5    1675      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-05013a81    161.2    1612      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-04013a80    0.0         -      -      -      -        -        -       -
 -       -
  i-3ab524a1    155.9    1559      0      0      0    0.003    0.002    0.001
 0.001   0.000
```

```
 i-bf300d3c    162.1    1621      0      0      0    0.003    0.002    0.001
0.001   0.000

 instance-id   type       az   running     load 1  load 5      user%  nice%
system%  idle%   iowait%
 i-d581497d    t2.micro   1a   25 mins       0.16     0.1        7.0    0.0
1.7   91.0       0.1
 i-d481497c    t2.micro   1a   25 mins       0.14     0.1        7.2    0.0
1.6   91.1       0.0
 i-136e00c0    t2.micro   1b   25 mins        0.0    0.01        0.0    0.0
0.0   99.9       0.1
 i-126e00c1    t2.micro   1b   25 mins       0.03    0.08        6.9    0.0
2.1   90.7       0.1
 i-8b2cf575    t2.micro   1c   1 hour        0.05    0.41        6.9    0.0
2.0   90.9       0.0

 instance-id   status     id   version              ago
      deployments
 i-d581497d    Deployed   2    v2                   9 mins
 i-d481497c    Deployed   2    v2                   7 mins
 i-136e00c0    Failed     2    v2                   5 mins
 i-126e00c1    Deployed   1    Sample Application   25 mins
 i-8b2cf575    Deployed   1    Sample Application   1 hour
```

上記の例は、5 つあるインスタンスのうち、3 番目のインスタンスでバージョン「v2」のデプロイに失敗した環境を示しています。デプロイの失敗後、想定されたバージョンは正常にデプロイされた最後のバージョン (この例では、最初のデプロイの「サンプルアプリケーション」) にリセットされます。詳細については「EB CLI を使用した環境ヘルスのモニタリング」を参照してください。

3. eb logs を実行して、アプリケーションのデプロイに関連付けられたログをダウンロードして表示します。

```
$ eb logs
```

4. eb ssh を実行して、アプリケーションを実行している EC2 インスタンスに接続し、直接調査します。インスタンスでは、デプロイされたアプリケーションは `/opt/python/current/app` ディレクトリに、Python 環境は `/opt/python/run/venv/` にあります。
5. eb console を実行して、AWS マネジメントコンソールでアプリケーション環境を表示します。ウェブインターフェイスを使用して、アプリケーションの設定、ステータス、イベント、ログな

ど、デプロイのさまざまな側面を簡単に調査できます。また、サーバーにデプロイした現在または過去のアプリケーションバージョンをダウンロードすることもできます。

Elastic Beanstalk のリソース

このサービスを利用する際に役立つ関連リソースは次のとおりです。

- Elastic Beanstalk API リファレンス すべての SOAP およびクエリ API の包括的な説明。さらに、すべての SOAP データ型のリストもあります。
- elastic-beanstalk-samples on GitHub – Elastic Beanstalk サンプル設定ファイル (.ebextensions) を含む GitHub リポジトリ。リポジトリの `README.md` ファイルには、サンプルアプリケーションが設定された追加の GitHub リポジトリへのリンクが含まれます。
- Elastic Beanstalk Technical FAQ – この製品について、開発者から寄せられるよくある質問。
- AWS Elastic Beanstalk リリースノート – Elastic Beanstalk サービス、プラットフォーム、コンソール、EB CLI リリースの新機能、更新、修正に関する詳細。

- クラスとワークショップ – AWS スキルを磨き、実践的な経験を積むために、セルフペースラボに加えて、ロールベースおよび専門コースへのリンク。
- AWS デベロッパーセンター – チュートリアルを試し、ツールをダウンロードして、デ AWS ベロッパーイベントについて学びます。
- AWS デベロッパーツール – AWS アプリケーションを開発および管理するためのデベロッパーツール、SDKs、IDE ツールキット、コマンドラインツールへのリンク。
- 入門リソースセンター – をセットアップし AWS アカウント、 AWS コミュニティに参加して、最初のアプリケーションを起動する方法について説明します。
- ハンズオンチュートリアル - ステップ バイ ステップのチュートリアルに従って、最初のアプリケーションを AWSで起動します。
- AWS ホワイトペーパー – ソリューションアーキテクトや他の技術専門家によって AWS 作成されたアーキテクチャ、セキュリティ、経済学などのトピックを網羅した、技術 AWS ホワイトペーパーの包括的なリストへのリンク。
- AWS サポート センター – AWS サポート ケースを作成および管理するためのハブ。フォーラム、技術的なFAQs、サービスのヘルスステータス、 など、その他の役立つリソースへのリンクも含まれています AWS Trusted Advisor。
- サポート – クラウドでのアプリケーションの構築と実行に役立つ サポート one-on-one の高速応答サポートチャネルである に関する情報のプライマリウェブページ。
- お問い合わせ - AWS の請求、アカウント、イベント、不正使用、その他の問題などに関するお問い合わせの受付窓口です。

- AWS サイト規約 – 当社の著作権と商標、お客様のアカウント、ライセンス、サイトアクセス、およびその他のトピックに関する詳細情報。

サンプルアプリケーション

以下は、Elastic Beanstalk の使用を開始する方法について説明します。 の一部としてデプロイされたサンプルアプリケーションへのダウンロードリンクです。

> **Note**
>
> 一部のサンプルでは、使用しているプラットフォームのリリース以降にリリースされた機能を使用する場合があります。サンプルの実行に失敗する場合は、「the section called “サポートされているプラットフォーム”」で説明されているように、プラットフォームを現行バージョンに更新してください。

- Docker - docker.zip
- 複数コンテナ Docker – docker-multicontainer-v2.zip
- 事前設定済み Docker (Glassfish) – docker-glassfish-v1.zip
- Go – go.zip
- Corretto – corretto.zip
- Tomcat – tomcat.zip
- Linux 上の .NET Core – dotnet-core-linux.zip
- .NET Core – dotnet-asp-windows.zip
- Node.js – nodejs.zip
- PHP – php.zip
- Python – python.zip
- Ruby – ruby.zip

AWS SDK for Java

AWS SDK for Java には、 AWS インフラストラクチャサービスを使用するアプリケーションを構築するために使用できる Java API が用意されています。を使用すると AWS SDK for Java、 AWS

Java ライブラリ、コード例、ドキュメントを含む単一のダウンロード可能なパッケージを数分で開始できます。

には J2SE 開発キット AWS SDK for Java が必要です。最新の Java ソフトウェアは、http://developers.sun.com/downloads/ からダウンロードできます。また、この SDK には Apache Commons (Codec、HTTPClient、および Logging) および Saxon-HE サードパーティパッケージも必要です。これらは、SDK の「third-party」ディレクトリに含まれています。

詳細については、AWS SDK for Java を参照してください。SDK、サンプルコード、ドキュメント、ツール、その他のリソースの詳細については、 AWS デベロッパーセンターの Java を参照してください。

AWS SDK for .NET

を使用すると AWS SDK for .NET、 AWS .NET ライブラリ、コード例、ドキュメントを含む 1 つのダウンロード可能なパッケージをすばやく開始できます。

詳細については、「AWS SDK for .NET」を参照してください。サポートされている .NET Framework および Visual Studio のバージョンの詳細については、AWS SDK for .NET デベロッパーガイドを参照してください。

SDK、サンプルコード、ドキュメント、ツール、その他のリソースの詳細については、 AWS デベロッパーセンターの .NET を参照してください。

AWS Toolkit for Visual Studio

AWS Toolkit for Visual Studio プラグインを使用すると、既存の .NET アプリケーションを Elastic Beanstalk にデプロイできます。で事前設定された AWS テンプレートを使用してプロジェクトを作成することもできます AWS SDK for .NET。

前提条件とインストールに関する詳細については、AWS Toolkit for Visual Studio を参照してください。Visual Studio を使用して Elastic Beanstalk アプリケーションを作成する場合は、「Elastic Beanstalk を使用した .NET Windows アプリケーションのデプロイ」を参照してください。

AWS Node.js での SDK for JavaScript

Node.js の AWS SDK for JavaScript を使用すると、 AWS Node.js ライブラリ、コード例、ドキュメントを含む単一のダウンロード可能なパッケージですぐに開始できます。

詳細については、Node.js 内のAWS SDK for JavaScript を参照してください。

AWS SDK for PHP

を使用すると AWS SDK for PHP、 AWS PHP ライブラリ、コード例、ドキュメントを含む 1 つのダウンロード可能なパッケージをすばやく開始できます。

詳細については、AWS SDK for PHP を参照してください。SDK、サンプルコード、ドキュメント、ツール、その他のリソースの詳細については、「 AWS デベロッパーセンター」の「PHP」を参照してください。

AWS SDK for Python (Boto)

を使用すると AWS SDK for Python (Boto)、Python ライブラリ、コード例、ドキュメントを含む 1 AWS つのダウンロード可能なパッケージをすばやく開始できます。API 上に Python アプリケーションを構築できるため、ウェブサービスのインターフェイスに対して直接コーディングをする複雑さがなくなります。

オールインワンのライブラリは、認証、再試行リクエスト、エラー処理を含む、 AWS クラウドのプログラミングに関連する低レベルのタスクを非表示にする Python のデベロッパーフレンドリーな API を提供します。ライブラリを使用したアプリケーションの構築方法については、SDK による Python の実用的な例が用意されています。

Boto の詳細情報、サンプルコード、ドキュメント、ツール、追加のリソースについては、AWS デベロッパーセンターの Python を参照してください。

AWS SDK for Ruby

AWS Ruby ライブラリ、コード例、ドキュメントを使用すると、Ruby アプリケーションを APIs 上に構築して、ウェブサービスインターフェイスに対して直接コーディングする複雑さを排除できます。

オールインワンのライブラリは、認証、再試行リクエスト、エラー処理を含む、 AWS クラウドのプログラミングに関連する低レベルのタスクを非表示にする Ruby のデベロッパーフレンドリーな API を提供します。ライブラリを使用したアプリケーションの構築方法については、Ruby による実用的な例が SDK に用意されています。

SDK の詳細情報、サンプルコード、ドキュメント、ツール、追加のリソースについては、AWS デベロッパーセンターの Ruby を参照してください。

AWS Elastic Beanstalk デベロッパーガイドのアーカイブされたコンテンツ

この章では、参照のために、いくつかの Elastic Beanstalk レガシー機能に関する設定情報を提供します。

トピック

- Elastic Beanstalk カスタムプラットフォーム (廃止)

Elastic Beanstalk カスタムプラットフォーム (廃止)

> **Note**
>
> 2022 年 7 月 18 日、Elastic Beanstalk は Amazon Linux AMI (AL1) に基づくすべてのプラットフォームブランチのステータスを廃止に設定します。これには、カスタムプラットフォームが含まれます。Elastic Beanstalk は、カスタムプラットフォームをサポートしていません。Elastic Beanstalk による Amazon Linux AMI の廃止に関する詳細については、「プラットフォームの廃止に関するよくある質問」を参照してください。

このトピックは、廃止前に Elastic Beanstalk カスタムプラットフォーム機能を使用したすべてのお客様の参照用として、このドキュメントに残ります。以前、Elastic Beanstalk カスタムプラットフォームでは、Amazon Linux AMI、RHEL 7、RHEL 6、または Ubuntu 16.04 ベース AMI からの AMI の構築をサポートしていました。これらのオペレーティングシステムは、Elastic Beanstalk によってサポートされなくなりました。サポートされなくなったカスタムプラットフォーム機能の詳細については、次のトピックを展開してください。

カスタムプラットフォーム

カスタムプラットフォームは、いくつかの点でカスタムイメージより高度なカスタマイズです。カスタムプラットフォームでは、ゼロから新しいプラットフォーム全体を開発し、プラットフォームインスタンスで Elastic Beanstalk が実行するオペレーティングシステム、その他のソフトウェア、およびスクリプトをカスタマイズできます。この柔軟性により、Elastic Beanstalk がマネージドプラットフォームを提供していない、言語やその他のインフラストラクチャソフトウェアを使用するアプリケーション用のプラットフォームを構築できます。既存の Elastic Beanstalk プラットフォームで使

用する Amazon マシンイメージ (AMI) を変更するカスタムイメージと比べた場合、Elastic Beanstalk は依然としてプラットフォームスクリプトを提供してプラットフォームのソフトウェアスタックを制御します。さらに、カスタムプラットフォームでは、自動のスクリプト化された方法を使用してカスタマイズを作成して維持します。一方、カスタムイメージでは、実行中のインスタンスで変更を手動で行います。

カスタムプラットフォームを作成するには、サポートされているオペレーティングシステムである Ubuntu、RHEL、または Amazon Linux (正確なバージョン番号については「Platform.yaml ファイルの形式。」の `flavor` エントリを参照) のいずれかから Amazon マシンイメージ (AMI) を作成し、それをカスタマイズします。Packer を使用して独自の Elastic Beanstalk プラットフォームを作成します。これは、Amazon Elastic Compute Cloud (Amazon EC2) で使用する AMI を含む多くのプラットフォームのマシンイメージを作成するためのオープンソースツールです。Elastic Beanstalk プラットフォームは、アプリケーションをサポートする一連のソフトウェアを実行するよう設定された AMI と、カスタム設定オプションやデフォルトの設定オプションの設定を含むことができるメタデータで構成されます。

Elastic Beanstalk は Packer を個別の組み込みプラットフォームとして管理するため、Packer の設定とバージョンについて心配する必要はありません。

Packer テンプレートと、AMI を構築するにテンプレートが呼び出すスクリプトおよびファイルを Elastic Beanstalk に提供して、プラットフォームを作成します。これらのコンポーネントは、テンプレートとメタデータを指定するプラットフォーム定義ファイルにより、プラットフォーム定義ファイルと呼ばれる ZIP アーカイブにパッケージ化されます。

カスタムプラットフォームの作成時に、Packer を実行する Elastic IP を使用しないで単一インスタンス環境を起動します。こうすることで Packer は別のインスタンスを起動してイメージを作成します。この環境は、複数のプラットフォームや各プラットフォームの複数のバージョンで再利用できます。

> **Note**
>
> カスタムプラットフォームは AWS リージョン固有です。複数のリージョンで Elastic Beanstalk を使用する場合は、各リージョンでプラットフォームを個別に作成する必要があります。
>
> 特定の状況下では、Packer で起動したインスタンスはクリーンアップされないため、手動で終了させる必要があります。これらのインスタンスを手動でクリーンアップする方法については、「Packer インスタンスのクリーンアップ」を参照してください。

アカウントのユーザーは、環境の作成中にプラットフォーム ARN を指定してカスタムプラットフォームを使用できます。ARN は、カスタムプラットフォームの作成に使用した eb platform create コマンドによって返されます。

カスタムプラットフォームを構築するたびに、Elastic Beanstalk は新しいプラットフォームのバージョンを作成します。ユーザーは、プラットフォームを名前で指定して最新バージョンのプラットフォームのみを取得したり、バージョン番号を指定して特定のバージョンを取得したりできます。

たとえば、最新バージョンのカスタムプラットフォーム (バージョン 3.0 など) を、ARN **`MyCustomPlatformARN`** を使用してデプロイする場合、EB CLI のコマンドラインは次のようになります。

```
eb create -p MyCustomPlatformARN
```

バージョン 2.1 をデプロイする場合、EB CLI のコマンドラインは次のようになります。

```
eb create -p MyCustomPlatformARN --version 2.1
```

カスタムプラットフォームバージョンの作成時にタグを適用できます。また、既存のカスタムプラットフォームバージョンのタグを編集できます。詳細については、「custom プラットフォームバージョンのタグ付け」を参照してください。

カスタムプラットフォームの作成

カスタムプラットフォームを作成するには、アプリケーションの root に `platform.yaml` ファイルを含める必要があります。このファイルは、カスタムプラットフォームの作成に使用されるビルダーのタイプを定義します。このファイルの形式は、「Platform.yaml ファイルの形式。」で説明しています。カスタムプラットフォームを最初から作成することも、サンプルカスタムプラットフォームのいずれかを開始点として使用することもできます。

サンプル カスタム プラットフォームの使用

独自のカスタムプラットフォームを作成する別の方法として、プラットフォーム定義アーカイブサンプルの 1 つを使用してカスタムプラットフォームをブートストラップできます。ソース AMI とリージョンを設定するだけで、サンプルを使用できます。

> **Note**
>
> 本番稼働で未変更のサンプルカスタムプラットフォームを使用しないでください。サンプルの目的は、プラットフォームで使用できる機能の一部を紹介することであり、本番稼働用としては強化されていません。

NodePlatform_Ubuntu.zip

このカスタムプラットフォームは、Ubuntu 16.04 に基づいており、Node.js 4.4.4 をサポートしています。このセクションの例では、このカスタムプラットフォームを使用しています。

NodePlatform_RHEL.zip

このカスタムプラットフォームは、RHEL 7.2 に基づいており、Node.js 4.4.4 をサポートしています。

NodePlatform_AmazonLinux.zip

このカスタムプラットフォームは、Amazon Linux 2016.09.1 に基づいており、Node.js 4.4.4 をサポートしています。

TomcatPlatform_Ubuntu.zip

このカスタムプラットフォームは、Ubuntu 16.04 に基づいており、Tomcat 7/Java 8 をサポートしています。

CustomPlatform_NodeSampleApp.zip

[express] と [ejs] を使用して静的なウェブページを表示する Node.js サンプル

CustomPlatform_TomcatSampleApp.zip

デプロイ時静的ウェブページを表示する Tomcat のサンプル

サンプルプラットフォーム定義アーカイブ `NodePlatform_Ubuntu.zip` をダウンロードします。このファイルには、プラットフォーム定義ファイル、Packer テンプレート、イメージの作成中に Packer が実行するスクリプト、およびプラットフォームの作成中に Packer がビルダーインスタンスにコピーするスクリプトと設定ファイルが含まれています。

Example NodePlatform_Ubuntu.zip

```
|-- builder                 Contains files used by Packer to create the custom platform
|-- custom_platform.json    Packer template
```

```
|-- platform.yaml           Platform definition file
|-- ReadMe.txt              Briefly describes the sample
```

プラットフォーム定義ファイル `platform.yaml` は、Elastic Beanstalk に Packer テンプレート (`custom_platform.json`) の名前を指定します。

```
version: "1.0"

provisioner:
  type: packer
  template: custom_platform.json
  flavor: ubuntu1604
```

Packer テンプレートは、HVM インスタンスタイプのプラットフォームイメージのベースとして Ubuntu AMI を使用して、プラットフォーム用の AMI を構築する方法を Packer に伝えます。`provisioners` セクションでは、アーカイブ内の `builder` フォルダのすべてのファイルをインスタンスにコピーし、インスタンスで `builder.sh` スクリプトを実行するよう Packer に伝えます。スクリプトが完了すると、Packer は変更したインスタンスからイメージを作成します。

Elastic Beanstalk は、Packer の AMI にタグを付けるために使用できる 3 つの環境変数を作成します。

AWS_EB_PLATFORM_ARN

カスタム プラットフォームの ARN。

AWS_EB_PLATFORM_NAME

カスタム プラットフォームの名前。

AWS_EB_PLATFORM_VERSION

カスタム プラットフォームのバージョン。

サンプル `custom_platform.json` ファイルは、これらの変数を使用して、スクリプトで使用する次の値を定義します。

- `platform_name` で設定されている `platform.yaml`
- `platform_version` で設定されている `platform.yaml`
- `platform_arn` は、メインビルドスクリプトによって設定され、`builder.sh` は、サンプル `custom_platform.json` のファイルの最後に表示される。

この `custom_platform.json` ファイルには、値を指定する必要がある 2 つのプロパティとして `source_ami` と `region` があります。AMI とリージョンの適切な値を選択する方法の詳細については、eb-custom-platforms-samples GitHub リポジトリの「Updating Packer template」(Packer テンプレートの更新) を参照してください。

Example custom_platform.json

```
{
  "variables": {
    "platform_name": "{{env `AWS_EB_PLATFORM_NAME`}}",
    "platform_version": "{{env `AWS_EB_PLATFORM_VERSION`}}",
    "platform_arn": "{{env `AWS_EB_PLATFORM_ARN`}}"
  },
  "builders": [
    {
      ...
      "region": "",
      "source_ami": "",
      ...
    }
  ],
  "provisioners": [
    {...},
    {
      "type": "shell",
      "execute_command": "chmod +x {{ .Path }}; {{ .Vars }} sudo {{ .Path }}",
      "scripts": [
        "builder/builder.sh"
      ]
    }
  ]
}
```

プラットフォーム定義アーカイブに含めるスクリプトとその他のファイルは、インスタンスに対して行う変更によって大きく異なります。サンプルプラットフォームには以下のスクリプトが含まれます。

- `00-sync-apt.sh` – コメントアウト: `apt -y update`。ユーザーに自動パッケージの更新を中断させるようプロンプトするため、このコマンドについてコメントアウトしました。これは、Ubuntu の問題の可能性があります。ただし、ベストプラクティスとして `apt -y update` を実行することを依然としてお勧めします。このため、このコマンドは参照用としてサンプルスクリプトに残してあります。

- `01-install-nginx.sh` – nginx をインストールします。
- `02-setup-platform.sh` – `wget`、`tree`、`git` をインストールします。フックとログ作成設定をインスタンスにコピーし、次のディレクトリを作成します。
  - `/etc/SampleNodePlatform` – ここに、デプロイ中にコンテナ設定ファイルがアップロードされます。
  - `/opt/elasticbeanstalk/deploy/appsource/` – ここに、`00-unzip.sh` スクリプトは、デプロイ中にアプリケーションのソースコードをアップロードします (このスクリプトについては Elastic Beanstalk 環境用のプラットフォームスクリプトツール セクションを参照)。
  - `/var/app/staging/` – ここで、アプリケーションのソースコードがデプロイ中に処理されます。
  - `/var/app/current/` – ここで、アプリケーションのソースコードが処理後に実行されます。
  - `/var/log/nginx/healthd/` – ここに、拡張ヘルスエージェントがログを書き込みます。
  - `/var/nodejs` – ここに、デプロイ中に Node.js ファイルがアップロードされます。

EB CLI を使用して、サンプルプラットフォーム定義ファイルで最初のカスタムプラットフォームを作成します。

カスタムプラットフォームを作成するには

1. EB CLI のインストール。
2. サンプルカスタムプラットフォームを展開するディレクトリを作成します。

   ```
   ~$ mkdir ~/custom-platform
   ```

3. `NodePlatform_Ubuntu.zip` をディレクトリに抽出し、展開されたディレクトリに移動します。

   ```
   ~$ cd ~/custom-platform
   ~/custom-platform$ unzip ~/NodePlatform_Ubuntu.zip
   ~/custom-platform$ cd NodePlatform_Ubuntu
   ```

4. `custom_platform.json` ファイルを編集し、`source_ami` プロパティと `region` プロパティの値を指定します。詳細については、「Updating Packer template」を参照してください。
5. eb platform init を実行し、プロンプトに従ってプラットフォームリポジトリを初期化します。

   eb platform を ebp に短縮することができます。

> **Note**
>
> Windows PowerShell では、コマンドエイリアスとして ebp を使用します。Windows PowerShell で EB CLI を実行している場合は、このコマンドの長い形式 eb platform を使用してください。

```
~/custom-platform$ eb platform init
```

このコマンドは、ディレクトリ `.elasticbeanstalk` を現在のディレクトリに作成し、設定ファイル `config.yml` をディレクトリに追加します。Elastic Beanstalk がカスタムプラットフォームを作成するときにこのファイルを使用するため、このファイルを変更または削除しないでください。

デフォルトでは、eb platform init は、現在のフォルダの名前をカスタムプラットフォームの名前として使用します。この例では、`custom-platform` です。

6. eb platform create を実行して Packer 環境を起動し、カスタムプラットフォームの ARN を取得します。この値は、後でカスタムプラットフォームから環境を作成する際に必要になります。

```
~/custom-platform$ eb platform create
...
```

デフォルトでは、Elastic Beanstalk はカスタムプラットフォームのインスタンスプロファイル `aws-elasticbeanstalk-custom-platform-ec2-role` を作成します。代わりに既存のインスタンスプロファイルを使用する場合は、eb platform create コマンドに `-ip` *`INSTANCE_PROFILE`* オプションを追加します。

> **Note**
>
> Elastic Beanstalk のデフォルトインスタンスプロファイル `aws-elasticbeanstalk-ec2-role` を使用すると、Packer によるカスタムプラットフォームの作成は失敗します。

EB CLI には、ビルドを完了するまでは Packer 環境のイベント出力が表示されます。Ctrl+C キーを押すと、イベントの表示を終了できます。

7. eb platform logs コマンドを使用してエラーがないか、ログを確認できます。

```
~/custom-platform$ eb platform logs
...
```

8. 後で、eb platform events によりプロセスを確認できます。

```
~/custom-platform$ eb platform events
...
```

9. eb platform status を使用してプラットフォームのステータスを確認します。

```
~/custom-platform$ eb platform status
...
```

オペレーションが完了すると、Elastic Beanstalk 環境の起動に使用できるプラットフォームが用意されます。

コンソールから環境を作成するときに、カスタムプラットフォームを使用できます。「新しい環境の作成ウィザード」を参照してください。

カスタムプラットフォームで環境を起動するには

1. アプリケーション用の新しいディレクトリを作成します。

```
~$ mkdir custom-platform-app
~$ cd ~/custom-platform-app
```

2. アプリケーションリポジトリを初期化します。

```
~/custom-platform-app$ eb init
...
```

3. サンプルアプリケーション NodeSampleApp.zip をダウンロードしてください。
4. サンプルアプリケーションの抽出

```
~/custom-platform-app$ unzip ~/NodeSampleApp.zip
```

5. eb create -p ***CUSTOM-PLATFORM-ARN*** (*CUSTOM-PLATFORM-ARN* は eb platform create コマンドで返された ARN) を実行して、カスタムプラットフォームを実行する環境を起動します。

```
~/custom-platform-app$ eb create -p CUSTOM-PLATFORM-ARN
...
```

プラットフォーム定義アーカイブのコンテンツ

プラットフォーム定義アーカイブは、アプリケーションソースバンドルのプラットフォームに相当します。プラットフォーム定義アーカイブは、プラットフォーム定義アーカイブ、Packer テンプレート、およびプラットフォームを作成するために Packer テンプレートによって使用されるスクリプトとファイルを含む ZIP ファイルです。

> **Note**
>
> EB CLI を使用してカスタムプラットフォームを作成する場合、EB CLI はプラットフォームレポジトリのファイルとフォルダーからプラットフォーム定義アーカイブを作成するため、アーカイブを手動で作成する必要はありません。

プラットフォーム定義ファイルは YAML 形式のファイルで、`platform.yaml` という名前でプラットフォーム定義アーカイブの root になければなりません。プラットフォーム定義ファイルでサポートされる必須キーおよびオプションのキーのリストについては、カスタムプラットフォームの作成を参照してください。

特定の方法で Packer テンプレートに名前を付けるせんが、ファイル名はプラットフォーム定義アーカイブで指定された provisioner テンプレートに一致する必要があります。Packer テンプレートを作成する手順については、公式の Packer ドキュメントを参照してください。

プラットフォーム定義アーカイブのその他のファイルには、AMI を作成する前にインスタンスをカスタマイズするためにテンプレートによって使用されるスクリプトとファイルがあります。

カスタムプラットフォームフック

Elastic Beanstalk は、カスタムプラットフォームで標準化されたディレクトリ構造をフックに使用します。これらはライフサイクルイベント中、および管理オペレーション (環境のインスタンスが起動された、ユーザーがデプロイを開始した、またはユーザーがアプリケーションサーバーの再起動機能を使用した) に応じて実行されるスクリプトです。

フックをトリガーするスクリプトを `/opt/elasticbeanstalk/hooks/` フォルダのサブフォルダの 1 つに配置します。

> **Warning**
>
> マネージド型プラットフォームでのカスタムプラットフォームフックの使用はサポートされていません。カスタムプラットフォームフックは、カスタムプラットフォーム用に設計されています。Elastic Beanstalk マネージド型プラットフォームでは、プラットフォームによって、動作が異なったり、問題が発生したりすることがあります。Amazon Linux AMI プラットフォーム (上記 Amazon Linux 2) は、場合によっては便利なため、注意した上で使用してください。
>
> カスタムプラットフォームフックは、Amazon Linux AMI プラットフォームに存在するレガシー機能です。Amazon Linux 2 プラットフォームでは、`/opt/elasticbeanstalk/hooks/` フォルダ内のカスタムプラットフォームフックは完全に終了しています。Elastic Beanstalk はそれらを読み込んだり実行したりしません。Amazon Linux 2 プラットフォームは、Elastic Beanstalk マネージドプラットフォームを拡張するために特別に設計された新しい種類のプラットフォームフックをサポートしています。カスタムスクリプトおよびプログラムは、アプリケーションソースバンドルのフックディレクトリに直接追加できます。Elastic Beanstalk は、さまざまなインスタンスプロビジョニング段階でそれらを実行します。詳細については、the section called “Linux プラットフォームの拡張” の「プラットフォームフック」セクションを参照してください。

フックは次のフォルダーに整理されます。

- `appdeploy` — アプリケーションのデプロイ中に実行されるスクリプト。Elastic Beanstalk は、新しいインスタンスが起動されたときと、クライアントが新しいバージョンのデプロイを開始したときに、アプリケーションのデプロイを実行します。
- `configdeploy` — クライアントがオンインスタンスのソフトウェア設定に影響する設定の更新 (たとえば、環境プロパティの設定や、Amazon S3 へのログローテーションの有効化など) を行ったときに実行されるスクリプト。
- `restartappserver` — クライアントがアプリサーバーの再起動オペレーションを行ったときに実行されるスクリプト。
- `preinit` — インスタンスのブートストラップ中に実行されるスクリプト。
- `postinit` — インスタンスのブートストラップの後で実行されるスクリプト。

`appdeploy`、`configdeploy`、および `restartappserver` フォルダには `pre`、`enact`、および `post` サブフォルダがあります。オペレーションの各段階で、`pre` フォルダのすべてのスクリプトがアルファベット順に実行され、次に `enact` フォルダ、`post` フォルダの順に実行されます。

インスタンスを起動すると、Elastic Beanstalk は `preinit`、`appdeploy`、`postinit` の順序で実行します。実行中のインスタンスへのそれ以降のデプロイで、Elastic Beanstalk は `appdeploy` フックを実行します。ユーザーがインスタンスソフトウェアの設定を更新すると、`configdeploy` フックが実行されます。`restartappserver` フックは、ユーザーがアプリケーションサーバーの再起動を開始するときのみ実行されます。

スクリプトでエラーが発生した場合、ゼロ以外のステータスで終了し、`stderr` に書き込んでオペレーションを失敗させることができます。`stderr` に書き込むメッセージは、オペレーションが失敗した場合に出力されるイベントに表示されます。Elastic Beanstalk は、この情報をログファイル `/var/log/eb-activity.log` に取り込み、オペレーションを失敗させたくない場合は 0 (ゼロ) を返します。`stderr` または `stdout` に書き込むメッセージはデプロイログに表示されますが、オペレーションが失敗しない限り、イベントストリームには表示されません。

Packer インスタンスのクリーンアップ

Packer ビルダープロセスを完了前に停止するなど特定の状況では、Packer によって起動されたインスタンスはクリーンアップされません。これらのインスタンスは Elastic Beanstalk 環境の一部ではなく、Amazon EC2 サービスを使用してのみ表示および終了できます。

手動でこれらのインスタンスをクリーンアップするには

1. Amazon EC2 コンソールを開きます。
2. Packer でインスタンスを作成したリージョンと同じ AWS リージョンにいることを確認します。
3. [Resources] (リソース) の下で *N* [Running Instances] (N 個の実行中のインスタンス) を選択します。*N* は実行中のインスタンスの数です。
4. クエリテキストボックスをクリックします。
5. [Name] タグを選択します。
6. 「packer」と入力します。

   クエリは次のようになります。[tag:Name: packer]
7. クエリに一致するインスタンスを選択します。
8. [Instance State (インスタンスの状態)] が [実行中] の場合は、[アクション]、[Instance State (インスタンスの状態)]、[Stop (停止)] の順に選択し、次に [アクション]、[Instance State (インスタンスの状態)]、[終了] の順に選択します。

Platform.yaml ファイルの形式。

`platform.yaml` ファイルの形式は次のとおりです。

```
version: "version-number"

provisioner:
   type: provisioner-type
   template: provisioner-template
   flavor: provisioner-flavor

metadata:
   maintainer: metadata-maintainer
   description: metadata-description
   operating_system_name: metadata-operating_system_name
   operating_system_version: metadata-operating_system_version
   programming_language_name: metadata-programming_language_name
   programming_language_version: metadata-programming_language_version
   framework_name: metadata-framework_name
   framework_version: metadata-framework_version

option_definitions:
   - namespace: option-def-namespace
     option_name: option-def-option_name
     description: option-def-description
     default_value: option-def-default_value

option_settings:
   - namespace: "option-setting-namespace"
     option_name: "option-setting-option_name"
     value: "option-setting-value"
```

プレースホルダーを該当する値に置き換えます。

version-number

必須。YAML 定義のバージョン。**1.0** を指定してください。

provisioner-type

必須。カスタム プラットフォームを作成するのに使用されるビルダーのタイプ。**packer** を指定してください。

provisioner-template

必須。*provisioner-type* の設定を含む JSON ファイル。

provisioner-flavor

オプション。AMI に使用する基本オペレーティング システム。次のいずれかです:

amazon (デフォルト)

Amazon Linux。指定されていない場合は、プラットフォームが作成された時点の Amazon Linux の最新バージョンです。

Amazon Linux 2 はサポートされているオペレーティングシステムフレーバーではありません。

ubuntu1604

Ubuntu 16.04 LTS

rhel7

RHEL 7

rhel6

RHEL 6

metadata-maintainer

オプション。プラットフォーム所有者の連絡先情報 (100 文字)。

metadata-description

オプション。プラットフォームについての説明 (2000 文字)。

metadata-operating_system_name

オプション。プラットフォームのオペレーティングシステムの名前 (50 文字)。この値は、ListPlatformVersions API の出力をフィルタリングするときに使用できます。

metadata-operating_system_version

オプション。プラットフォームのオペレーティングシステム のバージョン (20 文字)。

metadata-programming_language_name

オプション。プラットフォームがサポートするプログラミング言語 (50 文字)

metadata-programming_language_version

オプション。プラットフォームの言語のバージョン (20 文字)。

metadata-framework_name

オプション。プラットフォームで使用されるウェブフレームワークの名前 (50 文字)。

metadata-framework_version

オプション。プラットフォームのウェブフレームワークのバージョン (20 文字)。

option-def-namespace

オプション。`aws:elasticbeanstalk:container:custom` 下の名前空間 (100 文字)

option-def-option_name

オプション。オプション名 (100 文字)。プラットフォームがユーザーに提供する最大 50 個のカスタム設定オプションを定義します。

option-def-description

オプション。オプションについての説明 (1024 文字)。

option-def-default_value

オプション。ユーザーが指定しなかった場合に使用されるデフォルト値。

次の例では、オプション **NPM_START** を作成します。

```
options_definitions:
 -  namespace: "aws:elasticbeanstalk:container:custom:application"
    option_name: "NPM_START"
    description: "Default application startup command"
    default_value: "node application.js"
```

option-setting-namespace

オプション。オプションの名前空間。

option-setting-option_name

オプション。オプションの名前。Elastic Beanstalk によって提供されるオプションを 50 個まで指定できます。

option-setting-value

オプション。ユーザーが 1 つを指定しない場合に使用されるデフォルト。

次の例では、オプション **TEST** を作成します。

```
option_settings:
 - namespace: "aws:elasticbeanstalk:application:environment"
   option_name: "TEST"
   value: "This is a test"
```

custom プラットフォームバージョンのタグ付け

AWS Elastic Beanstalk カスタムプラットフォームバージョンにタグを適用できます。タグは、 AWS リソースに関連付けられたキーと値のペアです。Elastic Beanstalk リソースのタグ付け、ユースケース、タグのキーと値の制約、サポートされているリソースタイプの詳細については、「Elastic Beanstalk アプリケーションリソースのタグ付け」を参照してください。

カスタムプラットフォームバージョンの作成時にタグを指定できます。既存のカスタムプラットフォームバージョンでは、タグの追加や削除、既存タグの値の更新ができます。カスタムプラットフォームバージョンごとに最大 50 個のタグを追加できます。

カスタムプラットフォームバージョンの作成時にタグを追加する

EB CLI を使用してカスタムプラットフォームバージョンを作成する場合は。eb platform create の `--tags` オプションを使用してタグを追加します。

```
~/workspace/my-app$ eb platform create --tags mytag1=value1,mytag2=value2
```

AWS CLI または他の API ベースのクライアントで、 create-platform-version コマンドの `--tags`パラメータを使用してタグを追加します。

```
$ aws elasticbeanstalk create-platform-version \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --platform-name my-platform --platform-version 1.0.0 --platform-definition-bundle
 S3Bucket=amzn-s3-demo-bucket,S3Key=sample.zip
```

既存のカスタムプラットフォームバージョンのタグを管理する

既存の Elastic Beanstalk カスタムプラットフォームバージョンのタグを追加、更新、削除できます。

EB CLI を使用してカスタムプラットフォームバージョンを更新する場合は、 eb tags を使用してタグを追加、更新、削除、一覧表示します。

たとえば、次のコマンドでは、カスタムプラットフォームバージョンのタグを一覧表示します。

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:platform/my-platform/1.0.0"
```

次のコマンドでは、mytag1 タグを更新して mytag2 タグを削除します。

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:platform/my-platform/1.0.0"
```

オプションの完全なリストおよび詳細な例については、「eb tags」を参照してください。

AWS CLI または他の API ベースのクライアントでは、 list-tags-for-resource コマンドを使用してカスタムプラットフォームバージョンのタグを一覧表示します。

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:platform/my-platform/1.0.0"
```

カスタムプラットフォームバージョンのタグを追加、更新、または削除するには、 update-tags-for-resource コマンドを使用します。

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:platform/my-platform/1.0.0"
```

追加するタグと更新するタグを update-tags-for-resource の `--tags-to-add` パラメータで指定します。存在していないタグが追加され、既存のタグの値が更新されます。

> **Note**
>
> 一部の EB CLI および AWS CLI コマンドを Elastic Beanstalk カスタムプラットフォームバージョンで使用するには、カスタムプラットフォームバージョンの ARN が必要です。ARN を取得するには、次のコマンドを使用します。
>
> ```
> $ aws elasticbeanstalk list-platform-versions
> ```

出力をフィルタリングしてカスタムプラットフォームの名前に絞り込むには、 `--filters` オプションを使用します。

ドキュメント履歴

次の表は、「AWS Elastic Beanstalk デベロッパーガイド」に対する 2024 年 4 月以降の重要な変更点をまとめたものです。

変更	説明	日付
改訂トピック: とは AWS Elastic Beanstalk	わかりやすくするために、新しいアーキテクチャ図と概要をまとめました。	2025 年 6 月 5 日
トピックの改訂: EB CLI のセットアップ	要約および焦点を絞ったインストール手順。	2025 年 6 月 5 日
改訂トピック: PHP クイックスタート	アプリケーションを更新して再デプロイするステップを追加し、いくつかの手順を改善しました。	2025 年 6 月 5 日
トピックの改訂: 入門チュートリアル	わかりやすく、オンボーディングを迅速化するために、入門チュートリアルを合理化しました。	2025 年 6 月 5 日
再編成されたコンテンツ	重要な情報を強調表示し、検出可能性を向上させるためにコンテンツ構造を再編成しました。	2025 年 6 月 5 日
新しいトピック: EC2 セキュリティグループの管理	Elastic Beanstalk は、デフォルトの EC2 セキュリティグループから環境をオプトアウトするサポートを追加します。	2025 年 4 月 30 日
新しいトピック: IIS アプリケーションを Elastic Beanstalk に移行する	Elastic Beanstalk は、Windows IIS アプリケーションを移行する機能を追加	2025 年 4 月 18 日

	します AWS Elastic Beanstalk 。	
新しいトピック: AWS Secrets Manager および Parameter Store での Elastic Beanstalk AWS Systems Manager の使用	Elastic Beanstalk は、環境変数を使用してシー AWS Secrets Manager クレットと Parameter Store AWS Systems Manager パラメータを参照するサポートを追加します。	2025 年 3 月 31 日
AWS の `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` マネージドポリシーの更新。	AWS 管理ポリシーのアクセス許可を更新しました。	2025 年 2 月 27 日
起動テンプレートを優先して起動設定を廃止	このトピックを改訂して、Elastic Beanstalk 環境が Amazon EC2 Auto Scaling による起動設定の非推奨によって、起動テンプレートを優先してどのように影響を受けるかを説明しました。また、起動設定と起動テンプレートに関連する他のコンテンツも改訂しました。	2025年1月23日
新しいトピック: Windows プラットフォームブランチでの EC2 Fast Launch の使用	Elastic Beanstalk Windows プラットフォームリリースには、EC2 Fast Launch が有効になっているベース AMIs が含まれています。	2025年 1 月 22 日
新しいトピック: スポットインスタンス割り当て戦略	Elastic Beanstalk は、環境の作成中にスポット配分戦略設定のサポートを追加します。	2025年 1 月 15 日

AdministratorAccess-AWSElasticBeanstalk AWS 管理ポリシー	AWS 管理ポリシーのアクセス許可を更新しました。	2024 年 12 月 11 日
Graviton arm64 ファーストウェーブ環境の推奨事項をアーカイブされたコンテンツの章に	Graviton arm64 ファーストウェーブ環境の EB 推奨事項をアーカイブされたコンテンツの章に移動	2024 年 10 月 5 日
起動テンプレート	新しいトピック: 起動テンプレート。このトピックでは、Amazon EC2 Auto Scaling が起動テンプレートを優先し、起動設定を段階的に廃止している状況について説明します。	2024 年 10 月 1 日
Python の QuickStart	新しいトピック: Python の QuickStart	2024 年 9 月 24 日
制限付き VPC エンドポイントポリシーに必要な Amazon S3 バケットアクセス許可	新しいトピック: 制限付き VPC エンドポイントポリシーに必要な Amazon S3 バケットアクセス許可	2024 年 9 月 18 日
Tomcat での Java の QuickStart	新しいトピック: Tomcat で実行されている Java JSP アプリケーションの QuickStart	2024 年 9 月 12 日
Java の QuickStart	新しいトピック: Java の QuickStart	2024 年 9 月 12 日

Amazon EC2 Systems Manager を Docker プライベートリポジトリに使用する	復活トピック: Amazon EC2 Systems Manager を Docker プライベートリポジトリに使用する。トピックを置き換えます Elastic Beanstalk / SSM 統合がサポートされるまで、Docker プライベートリポジトリ AWS Secrets Manager の を使用します。	2024 年 9 月 8 日
EB CLI 2.6 (廃止) をアーカイブされたコンテンツの章に	EB CLI 2.6 (廃止) をアーカイブされたコンテンツの章に移動	2024 年 8 月 15 日
EB API (廃止) をアーカイブされたコンテンツの章に	EB API (廃止) をアーカイブされたコンテンツの章に移動	2024 年 8 月 15 日
Docker コンテナからの Elastic Beanstalk アプリケーションのデプロイ	更新と整理: Docker コンテナからの Elastic Beanstalk アプリケーションのデプロイ	2024 年 8 月 15 日
アーカイブされたコンテンツ	新しい章: アーカイブされたコンテンツ	2024 年 8 月 15 日
Windows ASP.NET の QuickStart	新しいトピック: Windows ASP.NET の QuickStart	2024 年 7 月 5 日
.NET Core on Windows の QuickStart	新しいトピック: .NET Core on Windows の QuickStart	2024 年 6 月 28 日
Docker の QuickStart	新しいトピック: Docker の QuickStart	2024 年 6 月 19 日
環境間の Amazon S3 バケットアクセスの防止	新しいトピック: 環境間の Amazon S3 バケットアクセスの防止	2024 年 6 月 12 日
.NET Core on Linux の QuickStart	新しいトピック: .NET Core on Windows の QuickStart	2024 年 5 月 28 日

PHP の QuickStart	新しいトピック: PHP の QuickStart	2024 年 5 月 10 日
Node.js の QuickStart	新しいトピック: Node.js の QuickStart	2024 年 5 月 5 日
Go の QuickStart	新しいトピック: Go の QuickStart	2024 年 5 月 5 日
Elastic Beanstalk プラットフォームのリリーススケジュール	今後のプラットフォームブランチリリース のスケジュールを含む新しいトピックを追加しました。廃止されるプラットフォームブランチのスケジュール と リタイアしたプラットフォームブランチの履歴 をこのトピックに移動しました。	2024 年 5 月 1 日
`AWSElasticBeanstalkRoleCore` AWS 管理ポリシー	AWS 管理ポリシーのアクセス許可を更新しました。	2024 年 4 月 30 日
`AWSElasticBeanstalkManagedUpdatesServiceRolePolicy` AWS 管理ポリシー	AWS 管理ポリシーのアクセス許可を更新しました。	2024 年 4 月 30 日
`AWSElasticBeanstalkManagedUpdatesInternalServiceRolePolicy` AWS 管理ポリシー	AWS 管理ポリシーのアクセス許可を更新しました。	2024 年 4 月 30 日
`AWSElasticBeanstalkMaintenance` AWS 管理ポリシー	AWS 管理ポリシーのアクセス許可を更新しました。	2024 年 4 月 30 日

AWSElasticBeanstalkInternalMaintenanceRolePolicy AWS 管理ポリシー	AWS 管理ポリシーのアクセス許可を更新しました。	2024 年 4 月 30 日

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。