

に充実していますが、AWS Shield Advancedの料金は年間3,000USDに加え、通信量に応じた金額が請求されるため、相応のコストがかかります。必ずしも有効化するのではなく、システムが取り扱うデータの重要度や保存先に応じて使い分けましょう。

●Amazon GuardDuty

Amazon GuardDutyは、ユーザの動作や通信をモニタリング・分析し、脅威を識別する脅威検出サービスです。GuardDutyは脅威インテリジェンス（脅威検知のインプットとなる情報）と機械学習モデルを用いており、継続的に進化しながら脅威検出の精度を高めています。AWSのセキュリティのベストプラクティスでは、GuardDutyは有効化することが推奨されています。

Amazon GuardDutyはVPCフローログ、CloudTrail、Route 53のDNSログをインプットして解析を行います。このとき、これらのログの設定をユーザ側で有効化する必要はありません。各サービスから取得した情報をもとに、Amazon GuardDutyは既知の脅威と未知の脅威を検出します。既知の脅威とは、脅威インテリジェンスに含まれた、既知の不正な操作や通信として登録されているものを指します。未知の脅威とは機械学習によって検出される異常な操作、通信パターンを指します。

●Amazon Inspector

RDSやLambdaなどのPaaSまたはサーバレスのサービスは、責任共有モデルにもとづきAWSの責任で脆弱性に対する対策が行われます。一方、EC2はOSの設定やソフトウェアの脆弱性管理はユーザの責任で実施する必要があります。

こうした脆弱性管理をサポートするサービスが、EC2の脆弱性を診断するAmazon Inspectorです。Amazon Inspectorでは、診断の基準となるルールパッケージを提供しており、このルールパッケージにもとづきEC2を評価します。ユーザはルールパッケージを複数組み合わせることで、より高度な脆弱性管理を実現できます。

Amazon Inspectorは大きく分けて2つの診断機能を有しています。外部ネットワーク型診断とホスト型診断です。

●ホスト型診断

ホスト型診断はサーバ内部から脆弱性を確認する診断方法です。Amazon Inspectorでは、EC2にAmazon Inspectorエージェントをインストールすることで診断を行います。エージェントはEC2からAmazon Inspectorにテレメトリ（インストール済みのパッケージ情報やソフトウェアの設定）を送信し、Amazon Inspectorは受け取った情報と選択されたルールパッケージにもとづいてEC2を診断します。診断結果はS3に保存できるほか、SNSでの通知にも対応しています。

●外部ネットワーク型診断

外部ネットワーク型診断は外部ネットワークから脆弱性を確認する診断方法です。最も一般的なネットワーク脆弱性診断の方法です。ホスト型診断と異なり、こちらの診断方法ではAmazon Inspectorエージェントの導入は不要です。

●AWS Systems Manager (Parameter Store)

Systems Managerは、AWSにおけるシステム運用で、ソフトウェアインベントリの収集やOSのパッチ適用、運用コマンド実行、環境変数の管理、セキュアなサーバアクセス、メンテナンス作業の自動化などを実行するサービスです。

ここでは、その機能の1つであるParameter Storeを紹介します。

「パラメータストア」は、AWSのサービスやアプリケーションで利用するパラメータを管理する機能です。KMSと統合されており、パスワードやDBの接続文字列などのセキュリティに関連するパラメータを暗号化してセキュアに管理することもできます。

パラメータストアはEC2やSystems Managerのほか、下記のサービスからもパラメータを参照できます。これらのサービスで利用するパラメータや環境変数をパラメータストアで一元管理することが可能となります※22。

- EC2
- ECS
- Lambda
- CloudFormation
- CodeBuild
- CodeDeploy
- CodePipeline

●AWS Secrets Manager

AWS Secrets ManagerはデータベースのパスワードやAPIキーなどデータ流出の危険性がある認証情報（シークレット）を集約し、管理するサービスです。アプリケーションはAWS Secrets Managerにアクセスすることでシークレットを取得できるため、シークレットをローカルに保持する必要がなくなります。また、シークレットはKMSで暗号化して保存することで、セキュアに管理できます。

前項で解説したAWS Systems Managerのパラメータストアでも、AWS Secrets Managerと同様に認証情報を一元管理できます。両者の違いとして挙げられるのが、AWS Secrets Managerによるシークレットの自動ローテーション機能です。

Secrets Managerは、シークレットの更新間隔を指定すると自動でパスワード変更を行います。この機能はRDS、Redshiftなどのデータベースサービスと統合されており、Secrets Manager内のDBのパスワードがローテーションされると自動的

※22 連携可能なサービスの詳細および最新情報は開発者ガイドを参照してください。https://docs.aws.amazon.com/ja_jp/systems-manager/latest/userguide/systems-manager-parameter-store.html

にデータベースサービス側のパスワードを変更します。パスワードローテーションの仕組みはSecrets Managerによって自動的に作成されるLambdaによって実現されます。このLambda関数を変更することで、APIキー以外のシークレットのローテーションも実機と連動させることができます。

●Amazon Detective

Amazon Detectiveは、各種AWSサービスから収集できるデータを分析、可視化し、インシデントの原因を特定するサービスです。Amazon DetectiveはAWS Security HubとAmazon GuardDutyと統合されており、ロールやAPI、インスタンス、ユーザを条件として、収集された情報をドリルダウン（収集された情報を掘り下げて情報を詳細化すること）できます。

●Amazon Macie

Amazon Macieは、S3バケットとS3バケット内のオブジェクトを分析し、機械学習とパターンマッチングを用いて、脅威の検出やデータ分類を行うサービスです。暗号化無効化や公開検知といったバケットの脅威を検出するだけでなく、オブジェクトを分類してクレジットカードや電話番号など個人情報や機密データを検出できます。Amazon GuardDutyと同様にAmazon Macieも機械学習を用いており、継続的な改善が行われています。

- ・2章9節「Amazon S3 - セキュリティ保護」にて、S3のセキュリティ対策全般について解説。
- ・2章9節「Amazon S3 - その他の機能」にて、CORSやサーバログアクセス機能について解説。
- ・4章1節「AWS CodeCommit - CodeCommitへアクセスする端末のセットアップ方法」にて、アクセス設定について解説。
- ・4章2節「Amazon ECR - ECRの接続」にて、アクセス設定について解説。
- ・5章2節「AWS CloudTrail - CloudTrailの機能」にて、イベントログの暗号化について解説。
- ・5章5節「Amazon SQS - SQSの利用」にて、メッセージの暗号化やアクセス制御について解説。
- ・5章7節「Amazon CloudFront - セキュリティとその他の機能」にて、通信の暗号化やアクセスポリシー等について解説。

3 各マネージドサービスのセキュリティに関するサポート

AWSが提供するマネージドサービスのセキュリティに関しては、本節で記述しているもの以外にも、別章のサービスの中でも解説しています（読みやすさの観点から、各サービスの解説は章ごとに1箇所にまとめています）。認定試験の対策で、AWSのセキュリティに関する知識を振り返って学習したい場合は、以下の章のセキュリティに関する記述も参考にしてください。

- ・1章3節「AWS開発の基本となるサービス - VPCのアクセス制御」にて、セキュリティグループやACLについて解説。
- ・2章1節「Amazon API Gateway - API Gatewayの重要な機能と特徴」にて、API Gatewayの認証・認可について解説。
- ・2章6節「ELB - ELBで共通する特徴」にて、SSL/TLSターミネーションについて解説。
- ・2章6節「ELB - ALB」にて、OIDCプロバイダ認証、Certificate Manager/WAFなどとの連携について解説。
- ・2章6節「ELB - NLB」にて、セキュリティグループ設定の考慮点やSSLパススルーパスについて解説。

演習問題

- 1** あるアプリケーションはSQSを使用して、キュー「MyQueue」にメッセージを送信する処理が実装されています。グループ「developers」に所属する、あるエンジニアが開発中にテストとして、マネジメントコンソール上から、キューにメッセージを送信しようとしたところ、メッセージが送信できませんでした。「MyQueue」には、以下のようなリソースベースポリシーが設定されています。メッセージが送信できない原因として考えられるものを選択してください。

```
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__owner_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "SQS:*",
      "Resource": "arn:aws:sqs:ap-northeast-1:123456789012:MyQueue"
    }
  ]
}
```

- A. このリソースポリシーはルートユーザしかSQSオペレーションが許可されていません。
- B. グループ「developer」にSQSオペレーションを許可する、アイデンティティベースポリシーが付与されていない可能性があります。
- C. パーミッションバウンダリーで、グループ「developer」にSQSオペレーションを禁止するポリシーが記述されている可能性があります。
- D. グループ「developer」にこのキューに対するオペレーションを禁止する、アイデンティティベースポリシーがアタッチされている可能性があります。

- 2** 開発チームは商用環境のEC2インスタンスにアプリケーションをデプロイしようとしています。このアプリケーションはS3にアクセスする必要があり、インスタンスにはIAMロール「application-role」を設定して起動します。アプリケーションはインスタンスプロファイルで「application-role」を使って、インスタンスマタデータから一時認証情報を取得し、S3へアクセスすることを想定します。ロールに設定するポリシーとして正しいものを選択してください。

- A. ロールの信頼ポリシーとして以下を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SampleTrustedPolicy",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "arn:aws:iam::123456789012:role/application-role"
      }
    }
  ]
}
```

3

- B. ロールの信頼ポリシーとして以下を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SampleTrustedPolicy",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      }
    }
  ]
}
```

C. ロールのアクセスポリシーとして以下を設定します。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Effect": "Allow",  
         "Action": "s3:GetObject",  
         "Principal": {"Service": "ec2.amazonaws.com"},  
         "Resource": "arn:aws:s3:::example_bucket/*"}  
    ]  
}
```

D. ロールのアクセスポリシーとして以下を設定します。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Effect": "Allow",  
         "Action": "s3:GetObject",  
         "Principal": {"Service": "arn:aws:iam::123456789012:role/application-role"},  
         "Resource": "arn:aws:s3:::example_bucket/*"}  
    ]  
}
```

3 開発チームは商用、ステージング、開発といった用途に沿ってアカウントを分けており、ステージング環境アカウントXXXXで使用する、テスト資材・データの取得のために、開発環境アカウントYYYYのS3のTestResourceBucketへ、ロールを使ってクロスアカウントアクセスしたいと考えています。各ロールに設定するポリシーのうち、誤っているものを選択してください。

A. ステージング環境のロールの信頼ポリシーとして以下を設定します。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Sid": "StagingRoleTrustPolicy",  
         "Effect": "Allow",  
         "Action": "sts:AssumeRole",  
         "Principal": {"AWS": "arn:aws:iam::YYYY:role/DevelopmentRole"}  
    ]  
}
```

B. 開発環境のロールの信頼ポリシーとして以下を設定します。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {"Sid": "DevelopmentRoleTrustPolicy",  
         "Effect": "Allow",  
         "Action": "sts:AssumeRole",  
         "Principal": {"AWS": "arn:aws:iam::XXXX:role/StagingRole"}  
    ]  
}
```

- C. ステージング環境のロールのアクセスポリシーとして以下を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAssumeCrossAccountRole",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::YYYY:role/DevelopmentRole"
    }
  ]
}
```

- D. 開発環境のロールのアクセスポリシーとして以下を設定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccountYYYYBucketAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::TestResourceBucket/*"
    }
  ]
}
```

- 4** ある企業では、オンプレミス環境にある Microsoft Active Directory で従業員のユーザデータが管理されており、この認証情報を使って、AWSリソースへアクセスさせたいと考えています。実現可能な構成のうち、最も適切なものを選択してください。

- A. SAML 2.0準拠のIDフェデレーションで実現します。IDプロバイダに対して認証を行った後、SAMLアサーションをユーザが受け取り、それをを使って、AssumeRoleWithSAML APIにより一時認証情報を取得して、AWSリソースへアクセスします。
- B. CognitoのIDプールで、SAML IDプロバイダを設定することで実現します。IDプロバイダに対して認証を行った後、GetIdでSAMLアサーションを取得して、AssumeRoleWithSAML APIにより一時認証情報を取得して、AWSリソースへアクセスします。
- C. CognitoのIDプールで、OIDCプロバイダを設定することで実現します。IDプロバイダに対して認証を行った後、GetIdおよびGetOpenIdToken APIでIDトークンを取得して、AssumeRoleWithWebIdentity APIにより一時認証情報を取得して、AWSリソースへアクセスします。
- D. カスタムIDブローカーアプリケーションで実現します。IDプロバイダに対して認証を行った後、ブローカーがAssumeRole APIにより一時認証情報を取得して、AWSリソースへアクセスします。

- 5** あるアプリケーションでは、S3に格納された画像や動画などのメディアコンテンツをクライアントへ提供しています。画像や動画は、クライアント固有のもので秘匿性が高く、またサイズが大きいものも含まれるため、クライアントからダイレクトにS3へアクセスする方式で実装したいと考えています。次の記述のうち正しいものを選択してください。

- A. STSでFederationToken APIを呼び出し、このトークンを使って、S3へアクセス可能な、有効期限がある署名付きURLを生成してクライアントへ返却します。
- B. STSでGetSessionToken APIを呼び出し、このトークンを使って、S3へアクセス可能な、有効期限がある署名付きURLを生成してクライアントへ返却します。
- C. STSでAssumeRole APIから、セッションポリシーを設定した一時認証情報を作成し、有効期限がある署名付きURLを生成してクライアントへ返却します。
- D. リソースベースポリシーとポリシー変数を利用して、ユーザごとに引き受けるロールを切り替えて、STSでAssumeRole APIから、一時認証情報を作成し、有効期限がある署名付きURLを生成してクライアントへ返却します。

6 ある企業で、新しいサービスを実現するためのアプリケーションを開発しており、画像を含む個人情報データをS3に保存することを検討しています。個人情報の保護には高いレベルのセキュリティが求められ、データを暗号化することはもちろん、暗号化・復号処理に対する監査ログを残すことも法律上求められています。このセキュリティ要件を満たす最も簡易な方法を選択してください。

- A. Amazon S3でバケットを作成する際に、SSE-S3による暗号化を有効化します。
- B. Amazon S3でバケットを作成する際に、SSE-KMSによる暗号化を有効化します。
- C. KMSカスタマーマスターkeyを指定して、クライアントサイドでファイルを暗号化してファイルをアップロードするSDK APIを使って、S3へファイルを送信します。
- D. KMS Encrypt APIで個人情報データを暗号化してから、S3へファイルアップロードします。

7 EC2で実行されるアプリケーションは、KMSを利用して、共有ストレージに出力するデータをエンベロープ暗号化により保護することを検討しています。正しい実装方法を選択してください。

- A. KMS GenerateDataKey APIでカスタマーマスターkeyを指定して、暗号化されたデータキーとブレーンなデータキーを取得します。暗号化されたデータキーで、出力するデータを暗号化し、この暗号化に使用したデータキーはその後ただちに破棄します。
- B. KMS GenerateDataKey APIでカスタマーマスターkeyを指定して、暗号化されたデータキーとブレーンなデータキーを取得します。ブレーンなデータキーで出力するデータを暗号化し、この暗号化に使用したデータキーはその後ただちに破棄します。
- C. KMS Encrypt APIでエンベロープ暗号化オプションを指定して、暗号化されたデータキーとブレーンなデータキーを取得します。暗号化されたデータキーで、出力するデータを暗号化し、この暗号化に使用したデータキーはその後ただちに破棄します。
- D. KMS Encrypt APIでエンベロープ暗号化オプションを指定して、暗号化されたデータキーとブレーンなデータキーを取得します。ブレーンなデータキーで出力するデータを暗号化し、この暗号化に使用したデータキーはその後ただちに破棄します。

8 仮想通貨取引に関する処理で使用する非対称キーをKMSで利用したいと考えています。EC2上でこれらのキーを使った処理を実現するために最も効率的な手法を選択してください。

- A. 中間CA証明書を発行するサードパーティから入手した非対称キーを、KMSでキーマテリアルインポートします。
- B. AWS SSE-KMS(サーバサイド暗号化)でKMS API経由で非対称キーを取得します。
- C. AWS CSE-KMS(クライアントサイド暗号化)でKMS API経由で非対称キーを取得します。
- D. AWS KMSでは非対称キーをサポートしていません。EC2上で OpenSSLなどのライブラリを使用して非対称キーを取り扱います。

9 商用環境でKMSのカスタマーマスターkeyを使用しているアプリケーションがあります。運用担当者が意識しておくべきキーの運用で誤っているものを選択してください。

- A. 使用されているキーのうちカスタマー管理CMKで、AWS KMSを使って作成したキーは、3年後にキーが自動ローテーションされます。
- B. 使用されているキーのうちカスタマー管理CMKで、キーマテリアルインポートしたキーは、1年後に手動でローテーションする必要があります。
- C. 使用されているキーのうちAWSマネージドCMKで、AWSマネージドサービスが使用しているキーは、3年後に自動ローテーションされます。
- D. 使用されているキーのうちAWS所有CMKは、AWSが所有および管理しているので、キーのローテーションを意識する必要はありません。

10 EC2上で実行されているアプリケーションで、CloudWatch LogsからKMS ThrottlingExceptionが発生したことを検知しました。原因として考えられるものを2つ選択してください。

- A. EC2上で実行しているアプリケーションの中で、SSE-KMSを使った暗号化を有効化しているマネージドサービスへのアクセス処理に問題がある。
- B. EC2上で実行している暗号化したEBSローカルストレージへの大量のアクセスが発生している。
- C. EC2上で実行されているアプリケーションの中で、SDKを使ったマネージドサービスへのアクセスでエクスプロンシャルバックオフが実装されていない可能性がある。
- D. KMS APIのリクエスト上限を超えるアクセスが瞬間的に発生した可能性がある。
- E. EC2上で実行されているアプリケーションの中で、KMS Encrypt APIの上限を超えたファイル数が取り扱われた可能性がある。

11 ある開発者が、何千人のユーザが利用するWebアプリケーションを設計しています。ユーザは自身のメールアドレスを使用して、サインアップしアプリケーションは各ユーザの属性を保存する必要があります。ユーザがWebアプリケーションのサインアップができるようにするにはどのサービスを利用するのが適切ですか。1つ選択してください。

- A. Amazon Cognito Sync
- B. Amazon Cognito ユーザプール
- C. Amazon Cognito ID プール
- D. Amazon AppSync

12 ある開発者が、金融のトランザクションと取引統計を表示するバンキング向けアプリケーションを作成しています。アプリケーションはログイン時のプロトコルとして多要素認証(MFA)を追加する必要があります。この要件を満たすためにどのサービスを利用すべきでしょうか。1つ選択してください。

- A. MFAが設定されたAWS IAMを利用します。
- B. MFAが設定されたAmazon Cognitoユーザプールを利用します。
- C. AWS Directory Serviceを利用します。
- D. MFAが設定されたAmazon CognitoID プールを利用します。

13 ある開発者が、エンターテインメント業界向けのモバイルアプリケーションを作成しています。アプリケーションは個人情報を含んでいるため、できるだけ安全にやりとりする必要があります。開発者が選択すべき認証フローの設定のうち、適切なものを2つ選択してください。

- A. Cognito管理者APIを使った認証(ALLOW_ADMIN_USER_PASSWORD_AUTH)で安全にサーバとのやりとりを行います。
- B. Lambdaトリガーによるカスタム認証(ALLOW_CUSTOM_AUTH)を有効にし、追加で検証処理を行います。
- C. クライアントシークレットを発行し、検証処理を追加します。
- D. クライアントシークレットを発行せず、認証フローを構成します。
- E. SRPプロトコルベースの認証(ALLOW_USER_SRP_AUTH)を有効にします。

14 ある組織がAmazon API Gatewayを通してAPIを提供するアプリケーションを開発しています。APIはAmazonやGoogle、Facebook、AppleなどのOpenIDプロバイダをベースとして認証され、APIはカスタムの認可モデルにもとづいてアクセス許可設定されなければなりません。APIの認証・認可モデルを構築するために最も簡単でセキュリティ性の高い設計方法はどれでしょうか。1つ選択してください。

- A. 認証と認可のためにAPIリクエストへ付与するクレデンシャルをAmazon ElastiCacheに保存します。
- B. Amazon DynamoDBを使ってクレデンシャルを保存し、AWS STSからの一時認証情報をアプリケーションに取得させます。認証と認可のためにクレデンシャルを設定してAPIをコールします。
- C. Amazon CognitoユーザプールとJWT(JSON Web Token)をベースとしたユーザ認証・認可のためのオーソライザを使用します。
- D. Amazon、FacebookおよびAppleのOpenIDトークンプロバイダーを構築します。ユーザはこのIDプロバイダを認証し、それぞれのAPIコールの認証にJWTを渡します。

15 あるWebサイトで、Amazon S3バケットに保存された画像を配信しています。このサイトはAmazon Cognitoの利用を可能にしており、ログインしていないゲストユーザがS3バケットから画像を表示できるようにする必要があります。開発者はどのようにしてゲストユーザがAWSリソースにアクセスできるようにすることができるか、正しいものを1つ選択してください。

- A. ユーザプールにブランクユーザIDを作成し、ユーザグループに追加して、AWSリソースへのアクセスを許可します。
- B. 新しいIDプールを作成し、認証されていないIDへのアクセスを有効にして、AWSリソースへのアクセスを許可します。
- C. 新しいユーザプールを作成し、認証されていないIDへのアクセスを有効にして、AWSリソースへのアクセスを許可します。
- D. ユーザプールにアノニマスユーザを作成し、アクセスを制限して、AWSリソースへのアクセスを許可します。

16 あるWebサイトはEC2インスタンス上にホスティングされています。月額の使用料金が、前月と比べて10倍以上上昇しています。考えられる問題と対応すべき対処のうち、誤っているものを選択してください。

- A. アクセスキーやシークレットキーが漏洩し、悪意のある攻撃者に不正利用されている可能性があります。キーをただちに無効化して再発行します。
- B. Webサイトで使用されているサーバミドルウェアの脆弱性を突いて、インスタンスマタデータから認証情報が漏洩し、悪意のある攻撃者に不正利用されている可能性があります。サーバミドルウェアを再点検し最新バージョンにアップデートします。
- C. Webサイトで使用されているサーバミドルウェアの脆弱性を突いて、インスタンスマタデータから認証情報が漏洩し、悪意のある攻撃者に不正利用されている可能性があります。EC2に割り当てられているIAMロールの権限を再点検し、必要最小限のものにアップデートします。
- D. キーペアが漏洩し、悪意のある攻撃者に不正利用されている可能性があります。キーペアをただちに無効化して再発行します。

17 あるアプリケーションはAWS LambdaからAmazon RDSデータベースを使用しています。セキュリティ要件として、RDSデータベースへの接続するユーザのパスワードは1か月でローテーションすることが求められています。要件を達成するために最も簡単な方法を選択してください。

- A. Secrets Managerを利用します。パスワード用のパラメータに対し、1か月ごとに自動ローテーションするよう設定します。
- B. Systems Manager Parameter Storeを利用します。パスワード用のパラメータに対し、Secure Stringオプションを有効にし、1か月ごとに自動ローテーションするよう設定します。
- C. AWS KMSを利用します。パスワード用のパラメータに対し、1か月ごとに自動ローテーションするよう設定します。
- D. AWS CloudHSMを利用します。パスワード用のパラメータに対し、1か月ごとに自動ローテーションするよう設定します。

18 ある開発者は開発中のシステムで採用するAWSセキュリティサービスを検討しています。提供されているサービスと機能の説明のうち正しいものを選択してください。

- A. DDoS攻撃を抑止するため、AWS WAFの利用を検討します。
- B. EC2の脆弱性を診断するため、Amazon Detectiveの利用を検討します。
- C. ユーザの動作や通信をモニタリング・分析し、脅威を識別するため、AWS GuardDutyの利用を検討します。
- D. 特定のIPアドレスからアクセスを抑止するため、AWS Shieldの利用を検討します。

解答

1 D ➔ 問題：258 ページ、本文解説：213-215 ページ

このポリシーは、SQSに設定されているリソースベースポリシーです。アクションの実行可否はリソースベースポリシーやアイデンティティベースのポリシーの組み合わせ方で決まります。特に明示的な拒否が1つでも含まれていると、その操作は実行不可となる点がポイントです。各選択肢の不正解理由は以下のとおりです。

- ・選択肢A：このプリンシパルの指定はルートユーザを指定しているわけではなくアカウント全体を設定するときに記載される方法です。アカウントARN(arn:aws:iam::123456789012:root)、または「AWS:」プレフィックスの後にIDを付けた短縮形を使用できます。
- ・選択肢B：リソースベースポリシーとアイデンティティベースポリシーはどちらかが許可されていれば、アクセス可能になります(OR条件)。問題のリソースベースポリシーはアカウント全体で「MyQueue」に関するSQSオペレーションがすべて許可されているので、考えられるのは明示的な拒否が存在する可能性です。
- ・選択肢C：パーミッションバウンダリーはグループに設定することはできません。ロールかユーザが対象です。

2 B ➔ 問題：259 ページ、本文解説：218 ページ

EC2インスタンスにIAMロールを設定して、インスタンスマタデータから一時認証情報を取得する場合、プリンシパルを「ec2.amazonaws.com」とする信頼ポリシーをロールに設定しておく必要があります。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：プリンシパルはロールを引き受ける、信頼されたエンティティを指定する必要があります。ここで、プリンシパルに自身のロールを設定するのは正しくありません。
- ・選択肢C、D：アクセスポリシーとしてプリンシパルを指定する必要はありません。また、ロールをプリンシパルに設定する場合の属性は「Service」ではなく、「AWS」です。

3 A ➔ 問題：261 ページ、本文解説：219-222 ページ

クロスアカウントでのロールの引き受けは、引き受ける対象のロール(この問題では開発アカウント)の信頼ポリシーとS3へのアクセスポリシーを、AssumeRole APIを呼び出すロール(この問題ではステージングアカウント)のアクセスポリシーにAssumeRole API実行権限を設定しておく必要があります。各選択肢の解説は以下のとおりです。

- ・選択肢A：これは不要な設定です。ステージング環境のロールは引き受けの対象ではありません。
- ・選択肢B：開発環境のロールはステージング環境のロールから引き受けられるため、信頼されたエンティティとして、プリンシパルにステージングのロールの設定が必要になります。
- ・選択肢C：開発環境のロールに対し、AssumeRole APIの実行権限が必要なため、このアクセスポリシーは必要です。
- ・選択肢D：開発環境のロールには、S3へのアクセス権限が必要になるため、このアクセスポリシーは必要です。

4 A ➔ 問題：263 ページ、本文解説：222-223 ページ

Microsoft Active Directoryは、SAML 2.0がサポートされたユーザディレクトリサービスのため、SAML 2.0準拠のIDフェデレーションで実現します。IDプロバイダに対して認証を行った後、SAMLアサーションをユーザが受け取り、それを使って、AssumeRoleWithSAML APIにより一時認証情報を取得してAWSリソースにアクセスします。他の選択肢の不正解理由は以下のとおりです。

- ・選択肢B：CognitoのIDプールでSAMLプロバイダを指定することはできますが、GetIdでSAMLアサーションを取得することはありません。このAPIはOIDCトークンをCognitoトークンに関連付ける場合に呼ばれるCognitoのAPIです。
- ・選択肢C：これはOIDCに準拠したIDプロバイダで有効な一時認証情報の取得方法です。
- ・選択肢D：これはSAMLやOIDCに準拠していない独自のIDストアを持つシステムで有効な一時認証情報の取得方法です。

5 C ➔ 問題：263 ページ、本文解説：228-232 ページ

STSを用いて、セッションポリシーを指定してアクセス対象のコンテンツのアクセス制御を詳細に設定することができます。コンテンツはクライアント固有のもので秘匿性が高いので、オブジェクト単位でアクセス制御を設定し、期限付きのURLでクライアントからアクセスさせるのがよい実装方式です。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：FederationToken APIで取得したトークンを使って、S3へアクセス可能な有効期限付き署名URLを生成することはできません。SDKで有効期限がある署名付きURLは生成できます。
- ・選択肢B：GetSessionToken APIで取得したトークンを使って、S3へアクセス可能な有効期限付き署名URLを生成することはできません。SDKで有効期限がある署名付きURLは生成できます。
- ・選択肢D：リソースベースポリシー変数を使って、アクセス可能な対象のバケットやオブジェクトキーをユーザごとに切り替えることはできますが、ユーザごとに引き受けるロールを切り替えることはできません。

6 B ➔ 問題：264 ページ、本文解説：238-240 ページ

SSE-KMSはS3バケット作成時に暗号化のデフォルトオプションとして選択できます。アップロード時にS3上で暗号化され、CloudTrailに暗号化の証跡がログとして蓄積されます。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：SSE-S3はS3バケット作成時に暗号化のデフォルトオプションとして選択できますが、デフォルトで暗号化に対する監査ログを残すことはありません。
- ・選択肢C：クライアントサイドでエンベロープ暗号化してファイルをS3へアップロードするオプションです。実装も容易で、データキーを生成するときKMS APIがコールされ、監査ログも証跡として記録されますが、アプリケーションがすべてこの方法で実装されていることを確認する必要があり、また、アプリケーションを介さずS3にアップロードされた場合の暗号化が考慮されていません。
- ・選択肢D：KMS Encrypt APIでデータを暗号化することはできます。実装も容易で、KMS APIがコールされる際に監査ログも証跡として記録されます。しかし、このAPIの暗号化するデータサイズ上限は4KB以下に制限されます。問題では画像を含むデータであり、大きめのファイルサイズとなることが想定されるので、この方法では実現が困難です。

7 B ➔ 問題：264 ページ、本文解説：235-237 ページ

KMS GenerateDataKey APIで生成した暗号化キーとプレーンデータキーを取得し、暗号化をプレーンデータキーで行います。また、暗号化に使用したプレーンデータキーはただちに破棄し、暗号化されたデータキーは復元のときに必要なため破棄してはいけません。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：KMS GenerateDataKey APIで生成した暗号化キーとプレーンデータキーを取得することは正しいですが、暗号化はプレーンデータキーで行う必要があります。また、暗号化されたデータキーは復元のときに必要なため破棄してはいけません。
- ・選択肢C、D：KMS Encrypt APIは暗号化を行うAPIであり、エンベロープ暗号化のためのデータキーを生成するオプションはありません。

8 C ➔ 問題：265 ページ、本文解説：234-235 ページ

CSE-KMSを使って、エンベロープ暗号化でのデータキーとしては対称キーと非対称キーを選択できます。非対称キーはKMS GenerateDataKeyPair APIを利用して、指定されたCMKから取得することができます。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：KMSのキーマテリアルインポートは非対称キーをサポートしません。対称キーのみです。
- ・選択肢B：サーバサイド暗号化はAWSマネジドサービスがKMSのキーを使用して暗号化する方式であり、非対称キーをユーザが選択して利用することはできません。
- ・選択肢D：2019年のアップデートでKMSは非対称キーをサポートしています^{※23}。

9 A ➔ 問題：265 ページ、本文解説：235 ページ

カスタマー管理CMKで、AWS KMSを使って作成したキーの自動ローテーション期間は1年です。

※23 <https://aws.amazon.com/jp/about-aws/whats-new/2019/11/aws-key-management-service-supports-asymmetric-keys/>

10 B、D

→ 問題：266 ページ、本文解説：240 ページ

暗号化されたEBSボリュームは、暗号化・復号が必要なタイミングでKMS APIにアクセスします。その過程でスロットルエラーが発生した可能性があります。もう1つ原因として考えられるものは、東京リージョンのKMS APIのリクエスト上限はデフォルトで10,000に設定されています。偶発的に処理が集中しスロットルエラーが発生した可能性があります。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：例えばS3などのSSE-KMSを使った暗号化を有効化しているマネージドサービスでスロットル超過エラーが発生する可能性はありますが、EC2のログに出力されることはありません。また、発生したとしても、マネージドサービスのアクセスが発生しないとスロットルエラーは起こりえません。
- 選択肢C：AWS SDKのマネージドサービスへのアクセスはエクスプロンシャルバックオフが組み込まれた状態で実装されています。
- 選択肢E：Encrypt APIは暗号化可能なファイルサイズが4KBに制限されていますが、暗号化するファイル数に上限はありません。ファイル数だけAPIを呼び出すだけです。なお、通常、暗号化は、エンベロープ暗号化で実装されるため、Encrypt APIの使用頻度はそんなに高くありません。

11 B

→ 問題：266 ページ、本文解説：241-242 ページ

何千ものユーザが利用するアプリケーションのユーザディレクトリサービスとしてCognitoユーザプールが適しています。ユーザプールでメールアドレスを利用したサインアップが可能です。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：Cognito Syncはさまざまなデバイスでデータを同期するのに使われるサービスです。なお、現在は、AppSyncの利用が推奨されています。
- 選択肢C：IDプールは認証されたユーザに対し、AWSリソースへの認可を与えるサービスです。
- 選択肢D：AppSyncは複数のデバイスでデータを同期するために使われるサービスです。

12 B

→ 問題：266 ページ、本文解説：241 ページ

多くのユーザが利用するアプリケーションのユーザディレクトリサービスとしてCognitoユーザプールが適しています。ユーザプールではMFAを追加することができます。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：IAMユーザにMFAを設定することはできますが、IAMユーザはユーザプールではなく、AWSアカウントに存在するユーザです。
- 選択肢C：AWS Directory ServiceはActive Directoryを管理するためのサービスです。
- 選択肢D：IDプールは認証されたユーザに対し、AWSリソースへの認可を与えるサービスです。

13 D、E

→ 問題：267 ページ、本文解説：243-244 ページ

モバイルアプリケーションで使用する場合は、より安全性の高いALLOW_USER_SR_P_AUTHの使用が推奨されます。また、クライアントシークレットはアプリクライアントの正当性を検証する場合のパスワードに相当するパラメータです。デバッグやデコンパイルなどにより、不特定多数の攻撃者に参照される可能性があるモバイルアプリケーションでの使用は推奨されません。その他の選択肢の不正解理由は以下のとおりです。

- 選択肢A：これは安全なサーバサイドのアプリクライアントで実装する方式です。
- 選択肢B：カスタム認証を追加して検証を行ってもよいですが、モバイルアプリケーションでの安全を保証するものではありません。
- 選択肢C：デバッグやデコンパイルなどを使用して、不特定多数に見られる可能性があるモバイルアプリケーションでクライアントシークレットを使用するべきではありません。

14**C**

➡ 問題：267 ページ、本文解説：242、245 ページ

API Gatewayでは、Cognitoを使用して、サードパーティのOpenIDプロバイダと連携し、JWTを使用したオーソライザを構築できます。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：API Gatewayは、デフォルトでElastiCacheを使用してOpenIDプロバイダのトークンを保存するオプションはありません。カスタムオーソライザで利用することはできますが、そのままでは安全な保存方法ではありません。
- ・選択肢B：API Gatewayは、デフォルトでDynamoDBを使用してOpenIDプロバイダのトークンを保存するオプションはありません。カスタムオーソライザで利用することはできますが、そのままでは安全な保存方法ではありません。
- ・選択肢D：OpenIDプロバイダのトークンブローカーとなる環境を構築することはできますが、自らAuthorization Serverを構築する必要があります。煩雑になります。

15**B**

➡ 問題：268 ページ、本文解説：248 ページ

Cognito IDプールには認証されていないゲストユーザーのアクセス権限を設定でき、これを使ってAWSリソースへのアクセスを許可します。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：CognitoユーザプールでブランクのユーザIDを作成することはできません。
- ・選択肢C：ユーザプールは認証のためのユーザディレクトリサービスであり、認可の仕組みは有していません。IDプールと連携する必要があります。
- ・選択肢D：アノニマス向けのユーザを作成することはできますが、不特定のリクエストをすべてアノニマスユーザー1つに結びつけることはCognitoにはできません。

16**D**

➡ 問題：268 ページ、本文解説：250 ページ

キーペアが漏洩すると、そのキーペアを設定しているEC2インスタンスが不正にアクセスされて、EC2メタデータなどから認証情報などがすでに漏洩している可能性があります。したがって、キーペアをただちに無効化して再発行するだけでは不十分です。認証情報を一新して環境を構築し直す必要があります。なお、本設問には盛り込んでいませんが、操作履歴を記録しておくためにCloudTrailを有効化しておくのもインシデント対応の基本手順の1つです。

17**A**

➡ 問題：269 ページ、本文解説：255 ページ

AWS Systems Manager Parameter Storeと、Secrets Managerは認証情報を一元管理できます。両者の機能的な違いとして挙げられるのが、AWS Secrets Managerではシークレットの自動ローテーションがサポートされていることです。Secrets Managerは、シークレットの更新間隔を指定すると自動でパスワード変更を行います。

18**C**

➡ 問題：269 ページ、本文解説：254 ページ

AWS GuardDutyはユーザの動作や通信をモニタリング・分析し、脅威を識別する脅威検出サービスです。その他の選択肢の不正解理由は以下のとおりです。

- ・選択肢A：これはAWS Shieldの説明です。
- ・選択肢B：これはAmazon Inspectorの説明です。
- ・選択肢D：これはAWS WAFの説明です。

第4章

展開関連サービス

4-1. AWS CodeCommit

4-2. Amazon ECR

4-3. AWS CodeBuild

4-4. AWS CodePipeline

4-5. AWS CodeDeploy

4-6. AWS CloudFormation

4-7. AWS Serverless Application Model

4-8. AWS Elastic Beanstalk

4-9. その他の展開関連サービス

4-1 AWS CodeCommit

AWS CodeCommitはGitベースのソースコードリポジトリサービスです。サーバ管理が不要で高い可用性を持っています。

1 CodeCommitの特徴

●完全マネージド型 Git ソースコードリポジトリ

CodeCommitは、AWSマネージドであるため、リポジトリの高可用性・保全性が担保されています。データのバックアップや、Gitリクエストを受け付けるAPIサーバのダウンを心配する必要はありません。Gitの標準機能をサポートしており、標準的なGitコマンドやツールが利用できます。また、GitHubをはじめとしたGitリポジトリと同じように、コミット履歴の表示やグラフの参照のほか、プルリクエストおよびファイル・行単位でのレビューコメントが追加できます。

●AWS マネージドサービスとの連携

CodeCommitと連携できるAWSサービスは多数用意されています。ここでは、代表的な例を紹介します。

【CodeCommitと統合できるその他のAWSサービス】

サービス	説明
Amplify	Amplifyは、モバイルアプリケーション向けのバックエンドを簡単に構築したり、フロントエンドのフレームワークを通じて高速な開発環境を提供するサービスです。AmplifyコンソールからCodeCommitリポジトリに接続することができます。
Cloud9	Cloud9は、クラウド上の統合開発環境（IDE）です。コードの作成・ビルド・テスト・デバッグ・リリースなどのリポジトリとしてCodeCommitが利用可能です。
CloudFormation	テンプレートを記述することで、CodeCommitリポジトリを含むAWSリソースを迅速に作成できます。
CloudTrail	CodeCommitに対して実行されたAPIコールやGitコマンドのイベントを記録し、ログをS3へ保存します。
CloudWatch Event	リポジトリをモニタリングし、イベントが発生すると、SQSやKinesis、Lambdaなどで処理を実行できます。
CodeBuild	クラウド環境における継続的インテグレーション実行のリポジトリとして、CodeCommitを利用できます。詳細は本章3節「CodeBuild」も併せて参照してください。

サービス	説明
CodeGuru Reviewer	CodeGuru Reviewerは機械学習を使用してソースコードの分析とコードレビューを行うサービスです。Reviewerの対象となる、CodeCommitのリポジトリを関連付け、プルリクエスト対象のコードを分析します。
CodePipeline	クラウド環境における継続的デリバリー実行のリポジトリとして、CodeCommitを利用できます。詳細は本章4節「CodePipeline」も併せて参照してください。
CodeStar	CodeStarはCodeBuildやCodePipelineを含む環境を典型的なテンプレートを通じて簡単に作成できるサービスです。資材リポジトリとして、CodeCommitを利用可能です。
Elastic Beanstalk	クラウド環境におけるPaaS環境の構築に、Elastic Beanstalkを利用できます。CodeCommitは資材のデプロイ用のリポジトリとして統合されています。
AWS KMS	KMSはデータ暗号化に使用されるサービスです（3章3節参照）。リポジトリの暗号化に使用されます。
AWS Lambda	リポジトリに発生したイベントをトリガーとしてLambda関数を実行できます。
Amazon SNS	リポジトリに発生したイベントをもとにトリガーを作成し、AWSの各種サービスやメールなどに通知できます。

●IAM ベースのアクセス制御

CodeCommitでは、リポジトリの作成・変更・削除を実行できるユーザを制御できるだけでなく、プランチレベルで操作するユーザを制御することもできます。これは、IAMポリシーの条件をベースとしたアクセス制御機能を利用してお、ユーザ/リポジトリ/プランチ単位で柔軟に操作を制限可能です。

2 CodeCommitへアクセスする端末のセットアップ方法

CodeCommitを利用する場合、接続するための認証情報を端末にセットアップする必要があります。ここでは代表的な方法を紹介します。

●HTTPS接続によるGit認証情報の設定

IAMで作成したユーザに対し、Gitの認証情報（ユーザ名とパスワード）を作成する、最も簡易的な方法です。HTTPSを用いるため、所属している組織でインターネットに接続する際にHTTP/HTTPSプロキシサーバを経由する必要がある場合や、統合開発環境（Integrated Development Environment : IDE）やHTTPS認証をサポートするサードパーティ製のGitのGUIツールを利用する場合は、こちらの方法でセットアップします。

●SSHプロトコルを使用したアクセス設定

ローカルコンピュータのSSH接続およびgitコマンドを使用したい場合の設定方法です。公開鍵/秘密鍵のペアを作成し、公開鍵をCodeCommitへアクセスするIAMユーザに設定します。ローカル端末ではオペレーティングシステムのSSHのアクセス設定にもとづき、秘密鍵を保存しておきます。CodeCommitのリポジトリに対し、使い慣れたgitコマンドでさまざまな操作を行う必要がある場合はこちらの設定を行っておくとよいでしょう。

●git-remote-codecommitを使用したアクセス設定

git-remote-codecommitはAWS Directory ServiceやActive Directory、Web IDプロバイダにあるID情報を使用したフェデレーテッド（統合）アクセスおよび一時認証を使用したい場合に有効なアクセス方法です。

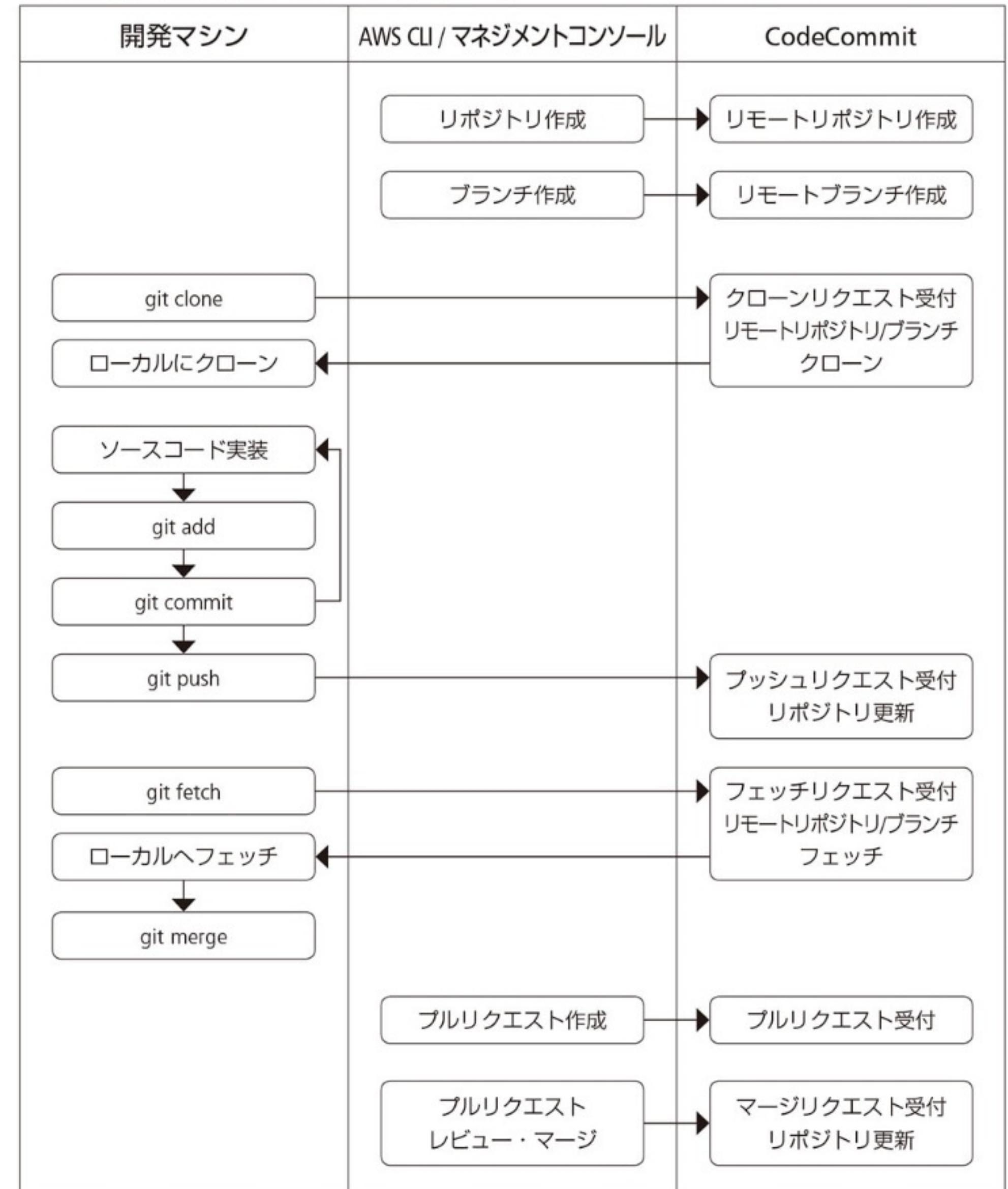
例えば、CodeCommitへのアクセス権限を付与したロールを作成しておき、端末にgit-remote-codecommitツールをインストールし、AWS CLIを使ってプロファイルやロールのARN（Amazon Resource Name）をセットアップします。以上で、Git認証情報や鍵の設定を行わずとも、端末のgitコマンドでリポジトリにアクセスできるようになります。

AWSでのほかのアカウントからのクロスアカウントアクセスや、既存のユーザディレクトリサービスのユーザを使用するなど、IAMユーザを作成したくない場合にもこちらの設定を用いるとよいでしょう。

3 CodeCommitの利用

CodeCommit（Git）では、以下のワークフローに従って、アプリケーションのソースコードを保存するリポジトリを作成し、開発を進めます。

【CodeCommitにおけるワークフロー^{※1}】



プルリクエストは特定のブランチでコミットした内容を別のブランチへマージ要求するためのリクエストです。

^{※1} https://docs.aws.amazon.com/ja_jp/codecommit/latest/userguide/welcome.html

バージョン管理したいファイルを、gitコマンドだけではなく、CLIやSDKを使ってCodeCommitに保存することもできます。CLIやSDKから実行できるAPIのうち、上記のフローで利用頻度が高いものは以下のとおりです。各APIの詳細については公式リファレンス^{※2}も参照してください。

【CLIやSDKで利用頻度の高いCodeCommit API】

カテゴリ	API	説明
リポジトリ操作	CreateRepository	CodeCommitリポジトリを作成します。
	DeleteRepository	CodeCommitリポジトリを削除します。
	ListRepositories	AWSアカウントに関連付けられたリポジトリの一覧を返します。
ブランチ操作	CreateBranch	指定されたリポジトリへブランチを作成します。
	DeleteBranch	リポジトリ内の指定されたブランチを削除します。
	GetBranch	指定されたブランチの情報を返します。
	ListBranches	指定されたリポジトリのすべてのブランチの一覧を返します。
	UpdateDefaultBranch	リポジトリにおけるデフォルトのブランチを変更します。
ファイル操作	DeleteFile	指定されたブランチからファイルを削除します。
	GetBlob	指定されたバイナリファイルをBase64エンコード形式で返します。
	GetFile	指定されたファイルをBase64エンコード形式で返します。
	GetFolder	指定されたディレクトリもしくはフォルダの中身を返します。
	PutFile	指定されたリポジトリおよびブランチ内の単一のファイルを追加もしくは修正します。
コミット操作	CreateCommit	リポジトリへの変更におけるコミットを作成します。
	GetDifferences	有効なコミット識別子の差分に関する情報を返します。

カテゴリ	API	説明
マージ操作	GetMergeCommit	変更前後のマージに関する情報を返します。
	GetMergeConflicts	プルリクエスト時に発生したマージの競合に関する情報を返します。
	MergeBranchesByFastForward	fast-forwardマージオプションを使ってマージします。
	MergeBranchesBySquash	squashマージオプションを使ってマージします。
プルリクエスト操作	MergeBranchesByThreeWay	ThreeWayマージオプションを使ってマージします。
	CreatePullRequest	指定されたリポジトリでプルリクエストを作成します。
	GetPullRequest	指定されたプルリクエストの情報を返します。
タグ操作	ListPullRequests	リポジトリにおけるすべてのプルリクエストの一覧を返します。
	ListTagsForResource	CodeCommit内で指定されたARN(Amazon Resource Name) で指定されたAWSタグの一覧を返します。
	TagResource	リソースにタグを付与したり、アップデートします。
	UntagResource	リソースのタグを消去します。



CLIやSDKを使って、CodeCommitへファイルを保存する方法を問われる場合があります。CodeCommitにおけるGitベースのワークフローやどのようなAPIが利用できるか、ひととおり押さえておくようにしましょう。

※2 https://docs.aws.amazon.com/ja_jp/codecommit/latest/APIReference/Welcome.html

4-2 Amazon ECR

「Amazon Elastic Container Registry (Amazon ECR)」は、AWSが提供する完全マネージド型のDockerコンテナレジストリで、コンテナイメージを保存・バージョン管理するサービスです。

1 ECRの概要

コンテナレジストリは、Docker Hubのようにクラウド上で提供されているものを利用したり、docker-distributionコマンドなどを用いてローカルサーバに構築したりすることもできます。ただし、Docker Hubの場合、無料で利用できるプルリクエスト数が限られており、特定ユーザに利用を絞るプライベートリポジトリは有料になります。一方で、ローカルにレジストリを構築する場合は、バックアップなどの保全性を担保する運用が大変です。

ECRは、コンテナイメージを非公開かつ、安全に保存でき、AWSネットワークからのプルリクエストは無料になります。また、保存したコンテナイメージを自動的に暗号化したり、プッシュしたイメージに脆弱性スキャンを施したりする無料のオプションがあります。レジストリ自体はAWSが冗長化構成で構築されているため、高可用性を担保しながら運用できます。また、4章で解説しているCodeBuildやCodePipelineといった継続的インテグレーション・デプロイのためのサービスなどと簡単に連携できるほか、第3章で解説したIAMポリシーを使って、アクセスするユーザをきめ細やかに制御することも可能です。

2 ECRの接続

ECRのレジストリに接続するには、CLI（コマンドラインインターフェース）から以下のとおり、ECRへログインするコマンドを実行します。

```
aws ecr get-login-password --region ap-northeast-1 | docker
login --username AWS --password-stdin <aws_account_id>.dkr.ecr.
ap-northeast-1.amazonaws.com
```

AWS CLIから直接ECRへログインするのではなく、AWS CLIから取得した認証情報を使って「docker login」コマンドによりECRレジストリへログインする点がポイントです。上記では「aws ecr get-login-password」コマンドから得られた認証情報を使って、「docker login」コマンドに渡しています。標準出力に認証情報を表示させるのではなく、パイプ（|）を使ってコマンド間で受け渡し、コマンド履歴に認証情報を残さないようにしておきましょう。

接続後はECRに対し、dockerコマンドを使って、イメージのプルやプッシュなどを実行することができます。

4-3 AWS CodeBuild

AWS CodeBuildは継続的インテグレーションの中核をなす要素です。ソースコードの変更を検知して、静的チェック・コンパイル・ユニットテストを実行し、不正を検知するために実行されます。

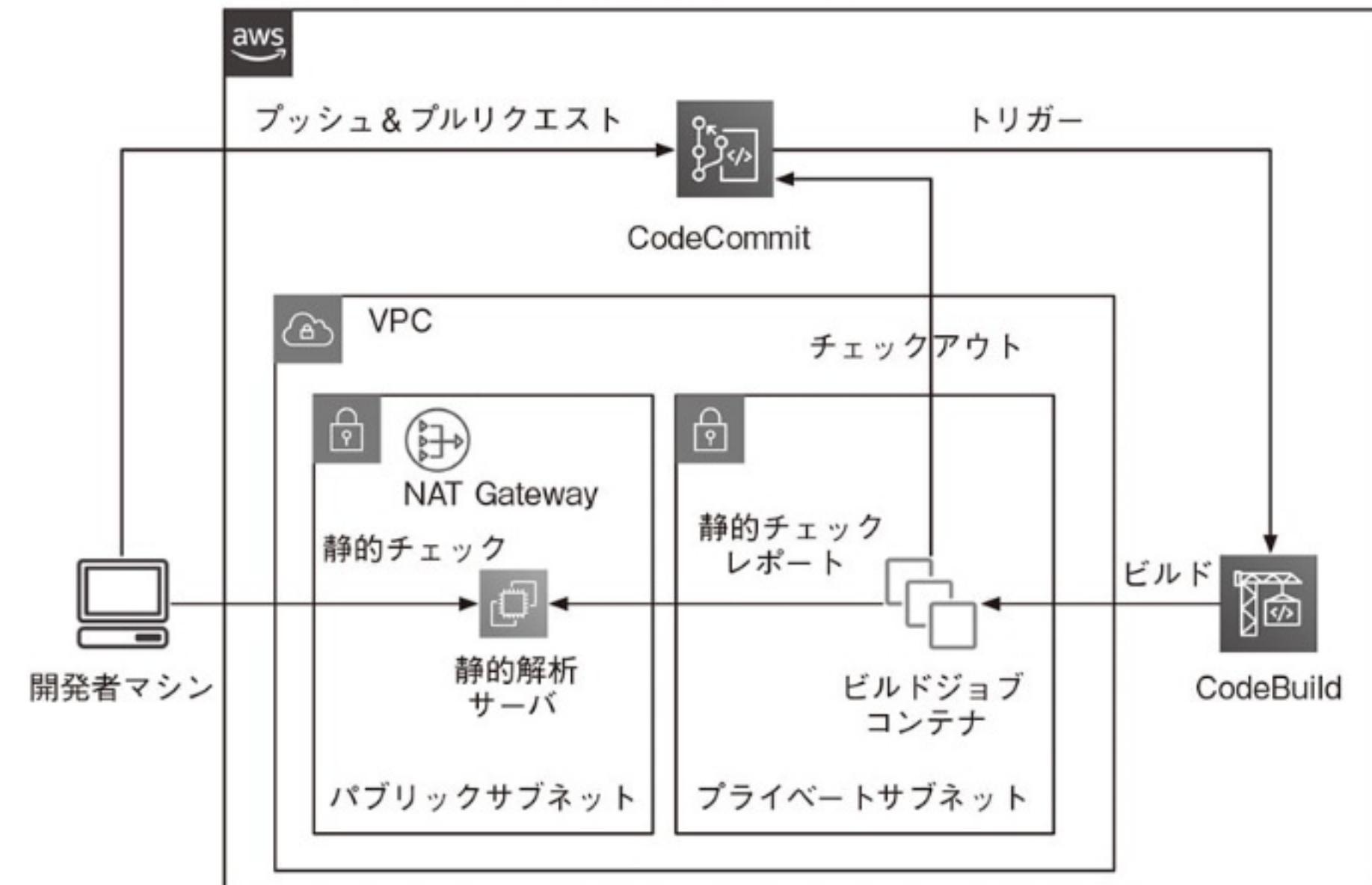
1 繼続的インテグレーションのコンセプト

CodeBuildについて説明をする前に、まず継続的インテグレーションが何を目指しているかを解説します。継続的インテグレーション（Continuous Integration : CI）は、リポジトリに蓄積されたコードの変更を検知し、開発環境での静的な解析やテスト、ビルトを実行し、バグを早期に発見してソフトウェアの品質を高める手法です。

次ページの図は、AWSで継続的インテグレーションを実現する環境の一例です。開発で中核となるアプリケーションの実装は主に開発者のマシンで行われ、適宜、静的解析サーバと連携しながら、静的チェックを行います。ある程度まとまった単位でリポジトリへプッシュし、ブランチをマージするプルリクエストが実行されますが、継続的インテグレーションでは、バグの混入を防ぐためこのタイミングでさまざまなチェックやテストを実行します。想定しない異常な結果が検出されれば、マージを失敗させたり、開発者や管理者に通知させたりすることで品質を確保しています。

こうした一連のフローの中で、CodeBuildはソースコードのビルドや静的チェック、コンパイル、ユニットテストなどを実行する位置付けとなるサービスです。類似のプロダクトとしてはオープンソースのCI/CDソフトウェア「Jenkins」が挙げられます。

【継続的インテグレーションの環境イメージ】



2 CodeBuildの特徴

CodeBuildではプロジェクトと呼ばれる単位で一連のビルドジョブを作成・実行します。ビルドの対象には、Amazon S3に保存したソースコードを指定できるほか、AWS CodeCommit、GitHub、BitBucketなどの各Gitベースのバージョン管理システムで扱うソースコードをサポートします。CodeBuildの特徴は以下のとおりです。

- ・ AWSのクラウド環境を利用してすることで、サーバリソースの制約を考えずビルド・テスト処理を実行できる
 - ・ ビルド処理はDockerコンテナイメージを使ったコマンドライン処理として実行される。ビルドに使うコンテナイメージは任意に指定できる
 - ・ 本章1節で解説したCodeCommitやCodePipeline（次節参照）とシームレスに連携し、簡単にビルドプロジェクトを構築できる
 - ・ Jenkinsとのプラグインを使ってCodeBuildと連携できる
 - ・ CloudWatch Logsにログを出力できる。また、ソースコードからビルドしたアプリケーションを、アーティファクトと呼ばれるアーカイブファイルとしてS3へ出力できる
 - ・ 機密データを管理するSystems Manager Parameter StoreやSecrets Managerと連携し、パラメータを環境変数経由で取得できる
 - ・ 任意の時間でプロジェクトを実行できるCodeBuildトリガーを利用できる

ビルドに使用されるコンテナイメージとして、さまざまなプログラミング言語のランタイムやブラウザのドライバ、Gitなど開発ツールがインストールされたAmazon Linux 2およびUbuntuがデフォルトで選択できます。

一連のビルドジョブはAWSのマネージドなサーバ環境下で行われますが、必要に応じてVPCに接続するように構成することもできます。その際、実行するサブネットにNAT Gatewayがアタッチし、アウトバウンド接続が許可されたセキュリティグループを設定する必要があります。また、ビルド実行するコンテナのCPU・メモリのセットを選択できるほか、実行スペックやビルドにかかるタイムアウト時間など細やかな設定が可能です。

3 buildspec.ymlの記述法

CodeBuildでは、ビルド処理をbuildspec.ymlというファイルに記述して、アプリケーションのソースコードプロジェクトのルートディレクトリに保存しておきます。環境変数やビルドに必要な環境のインストール処理、実際のビルド処理や生成するアプリケーションの出力などを定義します。サンプルは以下のとおりです。

```
version: 0.2
env:
  parameter-store:
    DOCKER_USER: "DOCKER_USER"
    DOCKER_PASSWORD: "DOCKER_PASSWORD"
    DOCKER_REPO : "DOCKER_REPO"
    IMAGE_REPO_NAME: "BACKEND_IMAGE_REPO_NAME"
    IMAGE_TAG: "BACKEND_IMAGE_TAG_STAGING"
phases:
  install:
    runtime-versions:
      docker: 18
  pre_build:
    commands:
      - echo Logging in to Docker Hub...
      - docker login -u $DOCKER_USER -p $DOCKER_PASSWORD $DOCKER_REPO
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG backend/build/production
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $IMAGE_REPO_NAME:$IMAGE_TAG
  post_build:
```

```
commands:
  - echo Build completed on `date`
  - echo Pushing the Docker image...
  - docker push $IMAGE_REPO_NAME:$IMAGE_TAG
  - printf '[{"name": "sample-continuous-delivery-backend-staging", "imageUri": "%s"}]' $IMAGE_REPO_NAME:$IMAGE_TAG > imagedefinitions.json
artifacts:
  files:
    - imagedefinitions.json
```

【buildspec.ymlで指定可能な代表的な要素】

要素		説明
version		buildspecのバージョンを指定します。0.1も利用可能ですが、要素名が変更されているため特別な理由がなければ最新バージョンを利用します。
env	variables	環境変数を指定します。
	parameter-store	SystemsManager Parameter Storeに保存されている環境変数を取得します。
phase	install	ビルドに必要なパッケージのインストール用途でコマンドを記述します。
	pre_build	リポジトリへのログインなど、ビルド実行前に必要な処理コマンドを記述します。
phase	build	ビルド処理コマンドを記述します。
	post_build	アーティファクトの整形やアップロードなどビルド後に実行する処理コマンドを記述します。
phase	run-as	コマンドを実行するユーザを指定します。実行コンテナをLinux系の環境とする場合のみ有効なオプションです
	on-failure	フェーズ中に障害が発生した場合の挙動をABORT/CONTINUEで設定します。ABORTの場合は処理を中断し、CONTINUEの場合は次のフェーズに進みます。
phase	runtime-versions	Ubuntu 標準イメージ2.0以降および Amazon Linux 2 標準イメージ1.0以降でサポートされているオプションです。各プログラミング言語やツールのビルドランタイムを指定できます。対象の言語はこちら ^{※3} を参照してください。
	commands	CodeBuildが実行する単一のコマンドを1つ以上記述します。複数ある場合はリスト順に実行されます。
phase	finally	commandsに記載された最後に実行されるコマンドです。commands内のコマンドが失敗した場合でも実行されます。
	artifacts	ビルド環境でのビルド出力アーティファクトのファイル名を記載します。

※3 https://docs.aws.amazon.com/ja_jp/codebuild/latest/userguide/build-spec-ref.html#build-spec-phases.install.runtime-versions

なお、buildspec.ymlを作成せずに、CodeBuildプロジェクト作成時に直接ビルドコマンドを指定することもできます。

4 その他の留意事項

CodeBuildを使用する際に、押さえておくべき留意事項として以下のようなものがあります。

- AWSはbuildspec.ymlの挙動をローカル端末で確認できる「CodeBuild Local」を提供しており、buildspec.ymlを簡単に検証できます。「CodeBuild Local」では、ビルドする環境となるコンテナイメージ^{※4}と、CodeBuildエージェントが含まれるコンテナイメージの2種類を準備します。ビルド環境となるコンテナイメージ、アーティファクトが出力されるアウトプットディレクトリを指定して、CodeBuild Local用のBashシェルスクリプト^{※5}を実行し、buildspec.ymlがエラーなく実行できるか検証します。
- buildspec.ymlでは、プロジェクトを同時実行するためのパッチビルドをオプションで記述できます。
- ビルドジョブを実行する際、アウトプットとしてDockerイメージを作成する場合は、特権付与（Privilegedモード）を選択しておく必要があります。
- buildspec.ymlは通常ソースコードのルートディレクトリに配置しますが、プロジェクト単位でファイルパスを指定でき、任意の場所に配置することも可能です。
- CodeBuildがSystems Manager ParameterStoreに保存されている環境変数を取得するには、CodeBuildに設定するサービスロールにssm:GetParametersアクションを追加する必要があります。

4-4 AWS CodePipeline

AWS CodePipelineは、ソースコードのコミット/プッシュやプルリクエストを契機として、テスト/ビルドやステージング/プロダクション環境へのデプロイといった一連のソフトウェアリリースプロセスを自動化/可視化し、継続的デリバリーを実現するサービスです。

1 継続的デリバリーのコンセプト

AWS CodePipelineについて説明をする前に、まず継続的デリバリーとは何かを説明します。

前節で解説した継続的インテグレーションが組み込まれた開発プロセスでは、ソースコードからアプリケーションがエラーなくビルドできることを保証されています。続いて、商用環境とほぼ同等のステージング環境へビルドされたアプリケーションをデプロイし、統合テストやユーザテスト、セキュリティを担保するペネトレーションテストをはじめ複数の試験を行います。試験をクリアし、品質が保証されたら、実際の商用環境へのデプロイを行います。この一連のプロセスを迅速かつ頻繁に実行可能にし、アプリケーションを隨時・確実にリリースできるようにすることが継続的デリバリーです。

継続的デリバリーには、最終的なデプロイに至るいくつかのステップがあり、そのうちのいくつかのプロセスは自動化されます。こうした一連の流れは、「パイプライン」とも表現されます。

【継続的デリバリーのパイプライン】



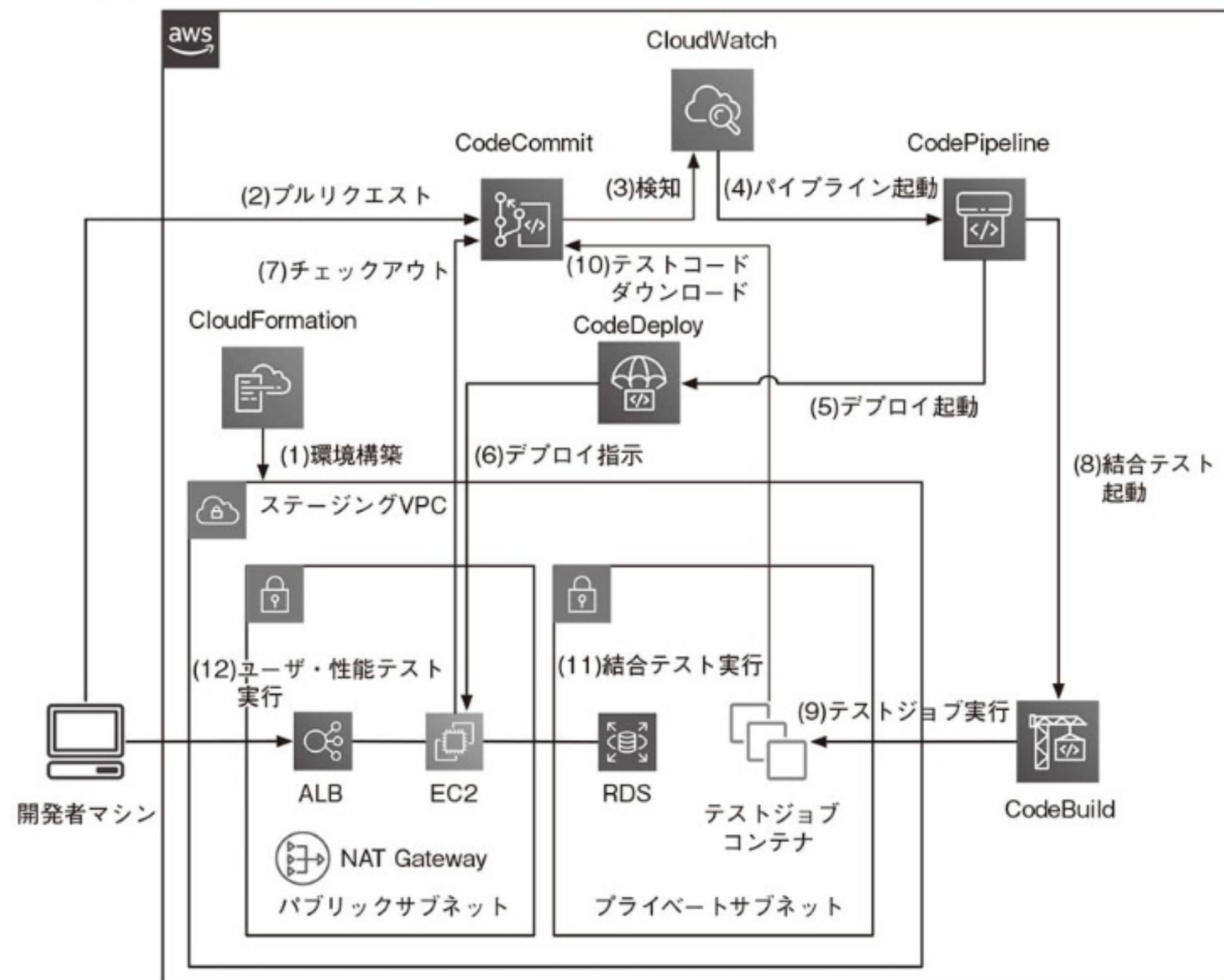
CodePipelineでは、これらのプロセスを自動化する

※4 <https://github.com/aws/aws-codebuild-docker-images>

※5 https://github.com/aws/aws-codebuild-docker-images/tree/master/local_builds

以下は前図のパイプラインに沿って、AWSで継続的デリバリーを実現するための環境の一例です。

【継続的デリバリーのパイプライン】



【継続的デリバリーのフロー】

No	フロー	説明
(1)	環境構築	開発者は基盤自動化サービスであるCloudFormationを使用して、ステージング環境を構築します。必要なときに環境構築することでコストを最適化できます。
(2)	プレリクエスト	アプリケーションをリリースしたいときは、リリース用のリポジトリのブランチにソースコードをプッシュしたり、コードオーナーへマージ要求のためのプレリクエストを発行します。ここでは、リポジトリにAWS CodeCommitを使用しています。
(3)	検知	リポジトリへの変更や要求を検知して、CloudWatch Eventへ通知します。
(4)	パイプライン起動	CloudWatch Eventが継続的デリバリーサービス・CodePipeline(次項参照)へ連携します。
(5)	デプロイ起動	CodePipelineがアプリケーションデプロイサービス「CodeDeploy(次節参照)」にデプロイを指示します。

No	フロー	説明
(6)	デプロイ指示	CodeDeployがデプロイを実行します。実際はデプロイ対象のサーバにインストールされたCodeDeployエージェントがデプロイ指示を検知して実行します。
(7)	チェックアウト	CodeDeployがCodeCommitからソースコードをダウンロードし、アプリケーション仕様(appspec)にもとづいて、アプリケーションをデプロイします。
(8)	結合テスト起動	CodePipelineがビルトサービス「CodeBuild」へテスト実行を指示します。
(9)	テストジョブ実行	CodeBuildがビルト仕様(buildspec)にもとづき、テストジョブを実行します。ここでは、ジョブはコンテナを使用したコマンドラインスクリプト処理として、指定されたVPCのサブネット内で実行するように設定しています。
(10)	テストコードダウンロード	実行されているジョブはリポジトリであるCodeCommitからテストコードをチェックアウトします。
(11)	結合テスト実行	チェックアウトした結合テストコードを実行します。
(12)	ユーザ・性能テスト実行	クライアントから本番を想定したユーザテストや性能テストなどを、手動もしくは自動化した手順で実行します。

フローの解説のとおり、環境構築やテストジョブ、デプロイといったパイプラインの各プロセスを実際に実行するのは、CloudFormationやCodeBuild、CodeDeploy、Elastic Beanstalkのようなサービスですが、CodePipelineはこれらパイプラインのプロセスフローを自動化・可視化し、管理する位置付けのサービスです。

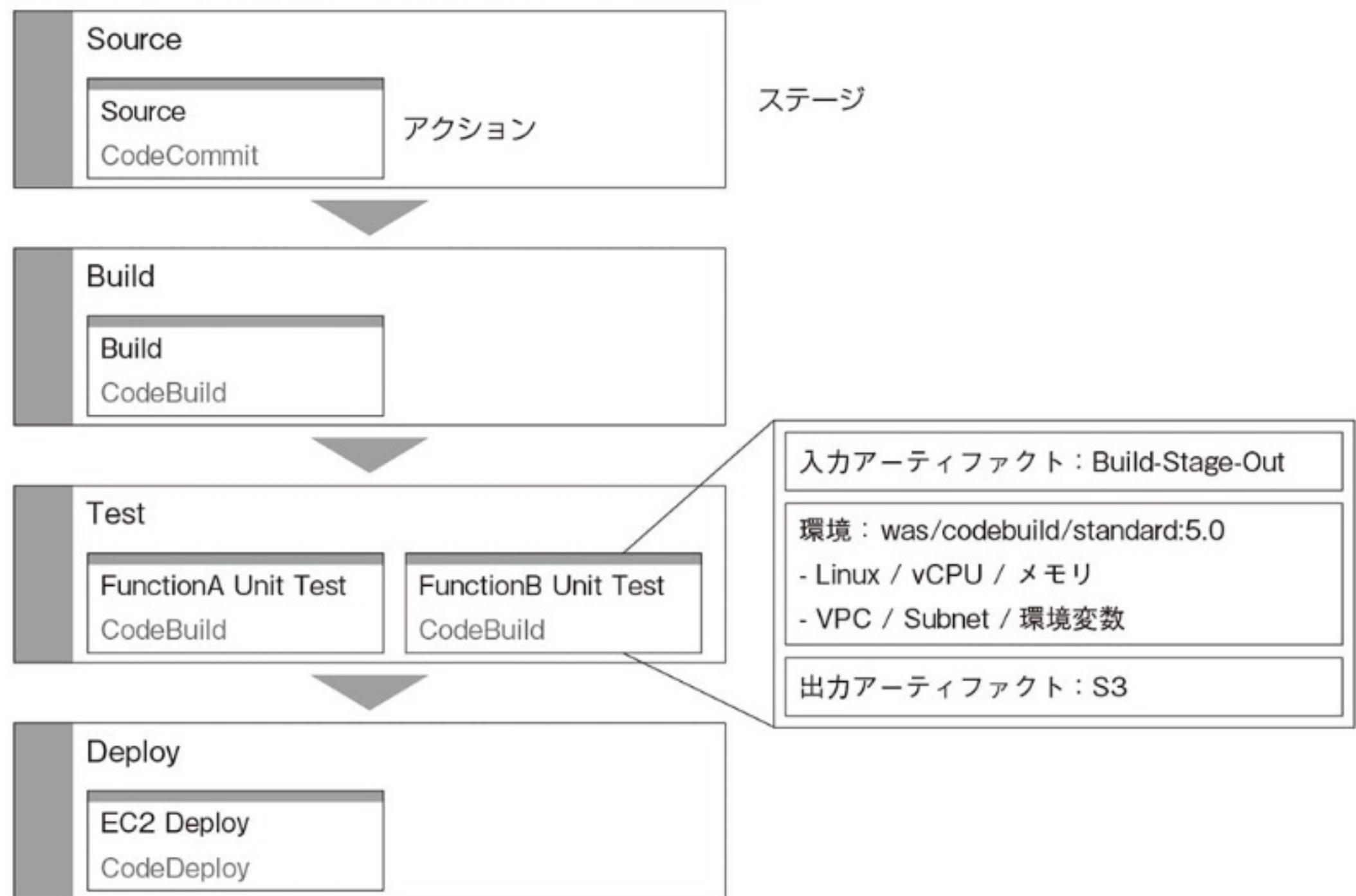
2 CodePipelineの特徴

CodePipelineでは、リリースまでのプロセスをパイプライン形式で定義します。主な特徴やメリットは以下のとおりです。

- ・AWSのさまざまなサービスと統合し、リリースプロセスを構築できる
- ・アプリケーション実行環境へのデプロイをシームレスに行える
- ・GitHubやJenkinsなどの主要なサードパーティリポジトリやCIツールとの連携もシームレスに実行できる

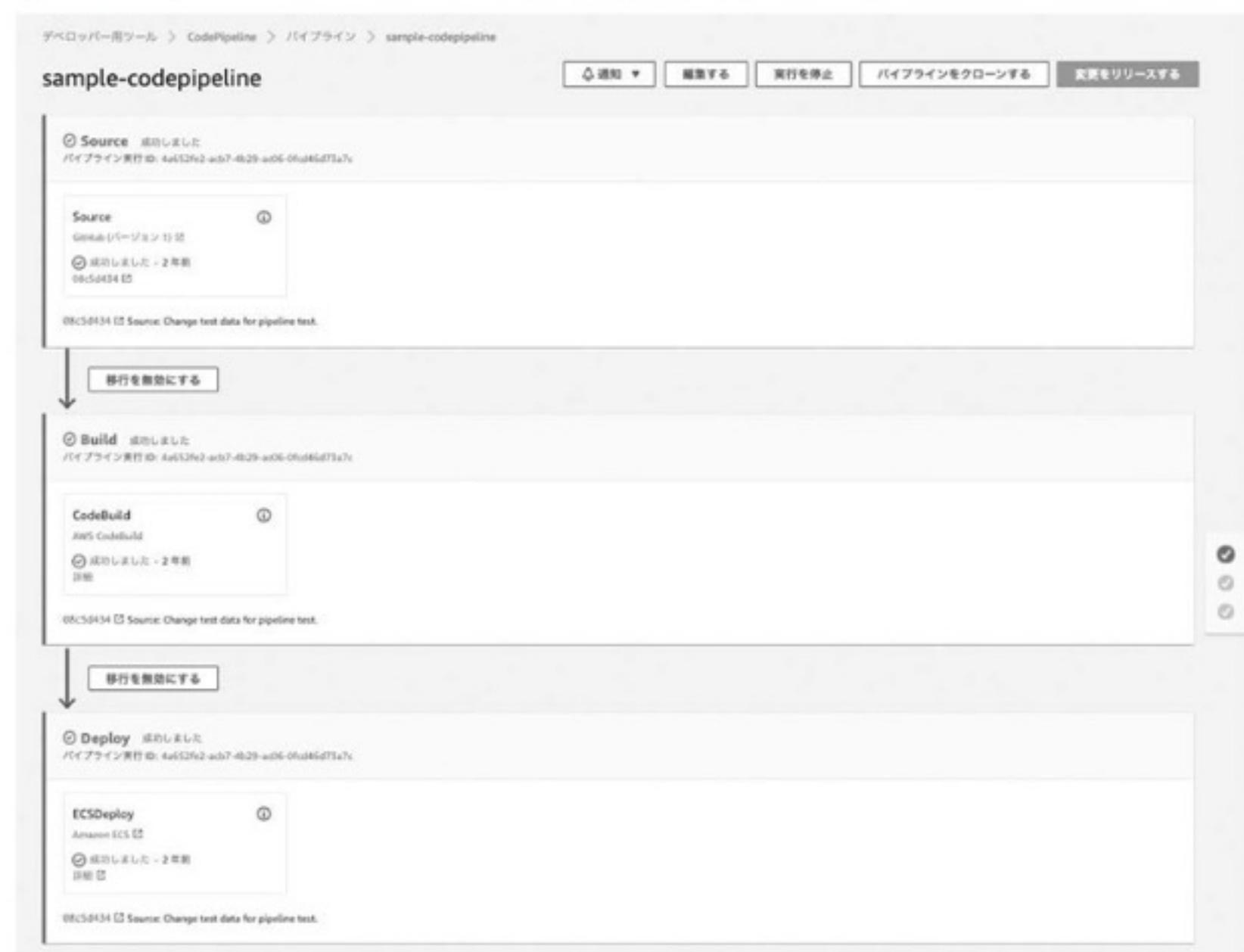
CodePipelineでは、以下のようにステージと呼ばれる単位で、それぞれ実行するアクションを定義してパイプラインを構築します。アクションはステージ内で逐次・並列に複数実行することも可能です。各アクションには、それぞれ実行環境や入出力のアーティファクトなど必要な要素を定義します。

【CodePipelineのステージとアクションの概念】



指定した条件でパイプラインが起動すると、各ステージの進行状況やアクションごとの実行履歴をコンソールでも確認できます。AWSコンソール（下図参照）では、パイプラインのステータスをリアルタイムに確認でき、トラブル発生時は状況に応じて素早く対応アクションをとることも可能です。

【CodePipelineのAWSコンソール上のイメージ】



CodePipelineでは、各ステージで以下のようなマネージドサービス、オープンソースツールやサードパーティを含むプロバイダとの連携をサポートしています。

【CodePipelineがサポートするプロバイダ^{※6}】

ステージ	説明	サポートするプロバイダ
Source	ソースコード・コンテナイメージを格納するリポジトリ	<ul style="list-style-type: none"> AWS CodeCommit Amazon ECR Amazon S3 GitHub GitHub Enterprise BitBucket GitLab
Build	ビルドジョブを実行するためのツール・サービス	<ul style="list-style-type: none"> AWS CodeBuild CloudBees Jenkins TeamCity
Test	テストジョブを実行するためのツール・サービス	<ul style="list-style-type: none"> AWS CodeBuild AWS Device Farm BlazeMeter GhostInspector MicroFocus StormRunner
Deploy	デプロイを実行するためのツール・サービス	<ul style="list-style-type: none"> Amazon S3 AWS AppConfig AWS CloudFormation Amazon ECS Elastic Beanstalk AWS OpsWorks AWS Service Catalog Amazon Alexa CodeDeploy XebiaLabs
Approval	承認を実行するための実行対象	<ul style="list-style-type: none"> Amazon SNS
Invoke	カスタム実行アクション	<ul style="list-style-type: none"> AWS Lambda AWS CodePipeline Snyk AWS Step Functions



CodePipelineとLambdaの関連付けの方法を問われる場合があるので、どういったかたちでCodePipelineからLambda関数の設定を行うのかイメージできるようにしておきましょう。詳細は4章2節を参照してください。

※6 https://docs.aws.amazon.com/ja_jp/codepipeline/latest/userguide/reference-pipeline-structure.html

3 その他の留意事項

CodePipelineを使用して以下のプロバイダを利用する際は、それぞれの留意点を押さえておきましょう。

- ・ソースコードプロバイダとしてCodeCommitを利用する場合では、ソースコード変更を検知するオプションとしてCloudWatch Eventsによる設定（推奨）と、CodePipelineによるポーリングが選択できます。
- ・ソースコードプロバイダとしてGitHubに接続する場合、OAuth 2.0を用いた認可によりアクセス制御されます。GitHub側のWebフック設定により、GitHubリポジトリに対するプッシュやプルリクエストなどのイベントが発生するとCodePipelineへ通知されます（推奨）。なお、CodePipelineによるポーリングも選択できます。
- ・ステージング環境などの構築を行う場合は、ソースステージの前にデプロイステージを差し込みます。デプロイステージのプロバイダにはCloudFormationを選択し、スタックの作成および更新オプションで実行するテンプレートを指定します。
- ・CodePipelineにはデプロイプロバイダとしてAmazon ECSが含まれており、実行中のアプリケーションコンテナをアップデートできます。アップデート方法には、Blue/Greenデプロイメントを選択できます。つまり、コンテナアプリケーションをデプロイする場合には必ずしもCodeDeployが必要なわけではありません。なお、ECSへコンテナをデプロイする場合は、サービスのコンテナ名およびイメージとタグを記述したJSONファイル（`imagedefinitions.json`）が必要になります。コンテナのリポジトリはこの記述ファイルで任意に指定できます。
- ・Approvalステージで選択できるSNSは、アクションを求めるコメントやレビュー用のURL、承認用のリンクを指定できます。
- ・カスタム実行アクションとして選択できるStepFunctionsを利用して、複雑なワークフローを持つリリースプロセスを構築できます。
- ・AWS CodePipelineはリージョンサービスのため、連携するプロバイダをVPC内のリソースに限定したり、VPCエンドポイントなどでVPC内から接続したりすれば、インターネットを経由することなくパイプラインを構築できます。
- ・プロダクション環境とステージング環境でアカウントを分ける場合は、S3およびKMSキーのクロスアカウントアクセスを有効化してパイプラインを構築します。これによりアカウントをまたいでアーティファクトを受け渡せるようになります。
- ・CloudWatch EventsやEventBridgeを使ってCodePipelineのイベントをモニタリングできます。キャンセル・失敗・再開・開始・停止・停止中・成功などのパイプラインの状態変化イベントに応じたアクションを設定できます。

4-5 AWS CodeDeploy

AWS CodeDeployはアプリケーションのデプロイに特化したサービスです。前節で解説したCodePipelineのデプロイステージから実行されるケースが多いサービスですが、オンプレミスからの実行など幅広い環境での利用が可能です。

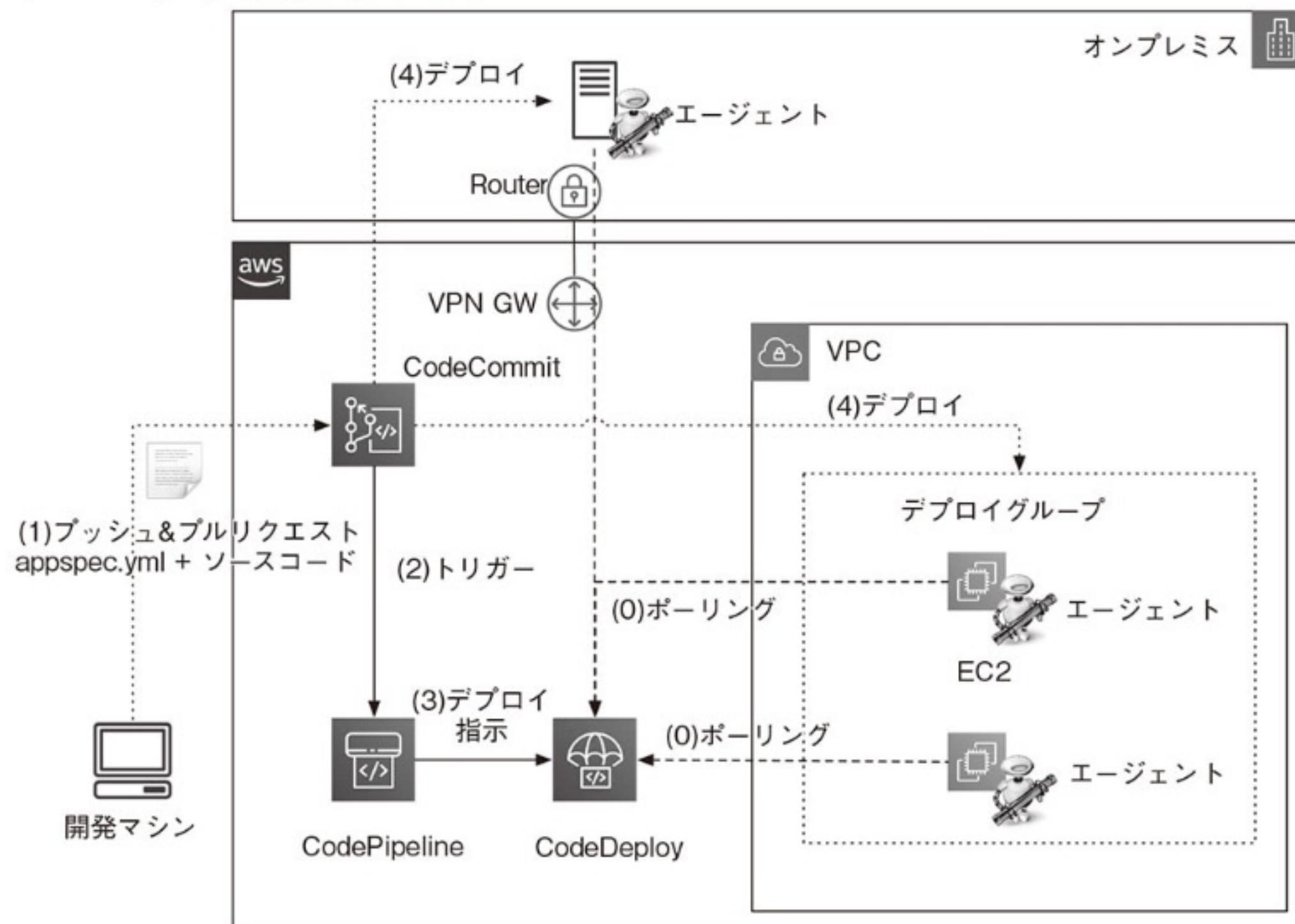
1 AWS CodeDeployの特徴

CodeDeployは、EC2およびオンプレミスサーバ、LambdaやECSをサポートしています。デプロイ対象となるインスタンスもしくはコンテナ（ノード）に新しいアプリケーションを置き換えていくIn-Placeデプロイメントと、新しいアプリケーションのノード群を作成し、テスト完了後にトラフィックを新しいノード群へ切り替えるBlue/Greenデプロイメントが選択できます。また、指定した複数のノードへ同時にデプロイを実行することも可能です。

CodeDeployでは、「デプロイグループ」というグループ単位で、デプロイ先のノードを扱います。デプロイグループは特定のタグを指定したEC2インスタンスもしくはAutoScalingグループから構成されます。次ページの図にCodeDeployが実行されるイメージを示します。

0. デプロイグループに含まれる各ノードには、事前にエージェントをインストールしておく
1. CodeCommitに対して、AppSpec（詳細は本節3項の「appspec.ymlの記述」で後述）を含むソースコード資材をプッシュまたはプルリクエストする
2. トリガー（CloudWatch Eventsがイベント検知）により、CodePipelineへ通知される
3. CodeDeployに対してデプロイ指示を行う
4. エージェントがこれを検知してソースコードやアプリケーションアーカイブを取得してデプロイ操作を実行する

【CodeDeployの実行イメージ】



CodeDeployを使ったデプロイ実行はエージェントからのポーリングにより実現されています。そのため、AWSクラウド内のリソースに限らず、サーバにエージェントをインストールしておけば、オンプレミス環境でもCodeDeployを使用できます。

なお、デプロイが失敗した場合、または監視しきい値が満たされた場合に、自動的にロールバックするようにデプロイグループまたはデプロイを設定することも可能です。



ユースケースや要件を満たすデプロイ設定を問われる場合があるので、次項を参考に、どのようなオプションを選択できるのか押さえておきましょう。

2 デプロイ設定

デプロイグループには、デプロイする割合や成功・条件を定義した「デプロイ設定」をアタッチします。デフォルトで用意されている代表的な「デプロイ設定」は以下のとおりです。なお、デプロイ設定はタイプや割合、間隔などを自分でカスタム定義することもできます。

【「デプロイ設定」でデフォルトで指定可能なデプロイ設定の代表例】

デプロイターゲット	タイプ/説明
EC2 /オンプレミス	CodeDeployDefault.OneAtTime 一度に1つのインスタンスにのみ新たなアプリケーションをデプロイする
EC2 /オンプレミス	CodeDeployDefault.HalfAtTime 一度に最大半分のインスタンスへ新たなアプリケーションをデプロイする(端数は切り捨て)
EC2 /オンプレミス	CodeDeployDefault.AllAtOnce 一度に可能な限り多くのインスタンスへ新たなアプリケーションをデプロイする
ECS	CodeDeployDefault.ECSEdge10PercentEvery1Minutes 1分ごとにトラフィックの10%をデプロイした新たなECSコンテナへシフトする
ECS	CodeDeployDefault.ECSCanary10Percent15Minutes 最初にトラフィックの10%を、15分後に残りの90%をデプロイした新たなECSコンテナへシフトする
ECS	CodeDeployDefault.ECSAllAtOnce すべてのトラフィックをデプロイした新たなECSコンテナへシフトする
Lambda	CodeDeployDefault.LambdaEdge10PercentEvery1Minute 1分ごとにトラフィックの10%をデプロイした新たなLambda関数へシフトする
Lambda	CodeDeployDefault.LambdaCanary10Percent15Minutes 最初にトラフィックの10%を、15分後に残りの90%をデプロイした新たなLambda関数へシフトする
Lambda	CodeDeployDefault.LambdaAllAtOnce すべてのトラフィックをデプロイした新たなLambda関数へシフトする

4



ユースケースや要件を満たすデプロイ設定を問われる場合があるので、どのようなオプションを選択できるのか押さえておくようにしましょう。

3 appspec.ymlの記述

エージェントがデプロイを行うためには、ソースコードのプロジェクトの配下に appspec.yml というファイルを配置しておく必要があります。appspec.yml はデプロイの仕様を定義したファイルで、ファイルの配置先や、デプロイのライフサイクルで実行するスクリプトなどを指定します。下記はアプリケーションサーバである Tomcat へアプリケーションアーカイブである war ファイルをデプロイする例です^{※7}。

```
version: 0.0
os: linux
files:
  - source: /target/SampleMavenTomcatApp.war
    destination: /tmp/codedeploy-deployment-staging-area/
  - source: /scripts/configure_http_port.xsl
    destination: /tmp/codedeploy-deployment-staging-area/
hooks:
  ApplicationStop:
    - location: scripts/stop_application
      timeout: 300
  BeforeInstall:
    - location: scripts/install_dependencies
      timeout: 300
  ApplicationStart:
    - location: scripts/write_codedeploy_config.sh
    - location: scripts/start_application
      timeout: 300
  ValidateService:
    - location: scripts/basic_health_check.sh
```

【appspec.ymlで指定可能な代表的な要素】

要素	説明
version	appspec のバージョンを指定します。現在は 0.0 のみが許容されています。必須パラメータです。
os	os 要素は EC2 およびオンプレミスサーバのみに有効なセクションです。デプロイ先のインスタンスでオペレーティングシステムの値を指定します。必須パラメータであり、「linux」か「windows」のいずれかを指定します。
files	files 要素は EC2 およびオンプレミスサーバのみに有効な要素です。
source	ソースコードリポジトリにあるコピー対象のファイルを指定します。
destination	コピー先のディレクトリを指定します。
resources	resources 要素は ECS および Lambda のみに有効な要素です。ECS と Lambda では使用できる子要素が異なります。
TaskDefinition	ECS のみに有効な要素で、必須パラメータです。デプロイする ECS タスク定義の ARN (Amazon Resource Name) を指定します。
ContainerName	ECS のみに有効な要素で、必須パラメータです。デプロイする ECS コンテナの名前を指定します。
ContainerPort	ECS のみに有効な要素で、必須パラメータです。デプロイする ECS コンテナがルーティングされるポートを指定します。
name	Lambda のみに有効な要素で、必須パラメータです。デプロイする Lambda 関数の名前を指定します。
alias	Lambda のみに有効な要素で、必須パラメータです。デプロイする Lambda 関数のエイリアスの名前を指定します。
currentversion	Lambda のみに有効な要素で、必須パラメータです。現在デプロイされている Lambda 関数のバージョンを指定します。
targetversion	Lambda のみに有効な要素で、必須パラメータです。デプロイする Lambda 関数のバージョンを指定します。
hooks	hooks 要素はすべてのデプロイ先で有効な要素ですが、デプロイ先によって子要素は異なります。デプロイサイクルイベントを表す子要素を任意に追加し、さらにその子要素として location 要素にスクリプト等のファイルを指定します。
ApplicationStop	この要素に指定したスクリプトは、デプロイアプリケーションのソースコードをダウンロードする前に実行されます。ただし、初回のデプロイ時は実行されません。既存のサーバで実行されているアプリケーションの停止スクリプトなどを実行する目的で指定します。

※7 <https://github.com/aws-samples/aws-codedeploy-sample-tomcat>

要素	説明
hooks	アプリケーションを一時的な場所にコピーする際に指定します。
	バックアップの作成などの用途で指定します。
	実際にアプリケーションをコピーする用途で指定します。
	アプリケーションをコピーした後の設定やアクセス権限の変更などを実施する用途で指定します。
	コピーしたアプリケーションを起動する用途で指定します。
	デプロイが正常に完了したことを確認する用途で指定します。
	ロードバランサーからの登録を解除する前の処理実行用途で指定します。
	ロードバランサーからの登録を解除した後の処理実行用途で指定します。
	ロードバランサーからの登録する前の処理実行用途で指定します。
	ロードバランサーからの登録した後の処理実行用途で指定します。
	ECSデプロイサイクル ^{※8} の中でイベントを表す子要素を任意に選択記述し、さらにその子要素に実行するLambda関数名を指定します。
	ECSタスクが置き換わる前の処理実行用途で指定します。
	ECSタスクが置き換えられた後の処理実行用途で指定します。
	テストリスナーがトラフィックを確認した後の処理実行用途で指定します。
Lambdaで有効な要素	2番目のターゲットグループのタスクが置き換わる前の処理実行用途で指定します。
	2番目のターゲットグループのタスクが置き換えられた後の処理実行用途で指定します。
	Lambdaデプロイサイクルの中でイベントを表す子要素を任意に選択記述し、さらにその子要素に実行するLambda関数名を指定します。
	Lambda関数が置き換わる前の処理実行用途で指定します。
BeforeAllowTraffic	Lambda関数が置き換えられた後の処理実行用途で指定します。

appspec.ymlは事前に作成せずとも、デプロイ設定時に直接AppSpecエディタを使用して記述することもできます。

※8 https://docs.aws.amazon.com/ja_jp/codedeploy/latest/userguide/tutorial-ecs-deployment-with-hooks.html

CodeDeployはEC2やオンプレミスなどのサーバ環境でのデプロイ用途に適しています。コンテナイメージ化したアプリケーションは、CodePipelineでECRなどのコンテナリポジトリから直接ECSへデプロイするオプションが選択できるので、単純なデプロイではCodeDeployを使用する必要はありません。デプロイグループにアタッチする条件を細かく設定したい場合などに使用を検討するとよいでしょう。

4 その他の留意事項

CodeDeployを使用する際に気をつけておきたい留意事項がいくつかあります。

- ・ オンプレミスでCodeDeployを使用する際は、以下の前提条件を満たす必要があります。
 - ・ エージェントのインストールおよび動作がテストされているオペレーティングシステムはAmazon EC2 AMI以外には、Ubuntuと、Red Hat Enterprise Linux、Windowsサーバ^{※9}です。それ以外はサポート外となります。
 - ・ CodeDeployエージェントはポート443を使用して、アウトバウンド通信します。オンプレミスネットワークからAWSのネットワークへ接続が許可されている必要があります。
 - ・ システムの構成エージェントの実行はsudoまたはroot、管理者権限が必要になります。
 - ・ オンプレミスインスタンスをCodeDeployサービスに登録するために、IAM ID（オンプレミスサーバを登録するユーザまたはロール）に対し、適切なアクセス許可を付与する必要があります。
 - ・ CodeDeployにはデプロイ対象に応じて適切な管理ポリシーを以下のように割り当てる必要があります。
 - ・ AWSCodeDeployRole
 - ・ AWSCodeDeployRoleForLambda
 - ・ AWSCodeDeployRoleForLambdaLimited
 - ・ AWSCodeDeployRoleForECS
 - ・ AWSCodeDeployRoleForECSLimited
 - ・ VPCエンドポイントを介することで、インターネットにアクセスすることなくVPCからCodeDeployにアクセスできます。ただし、使用するサービスに応じて以下のエンドポイントを指定しておく必要があります。
 - ・ CodeDeployからEC2へデプロイする場合は、codedeployおよびcodedeploy-commands-secureエンドポイント
 - ・ LambdaやECSへのデプロイする場合は、codedeployエンドポイント

※9 https://docs.aws.amazon.com/ja_jp/codedeploy/latest/userguide/codedeploy-agent-html#codedeploy-agent-supported-operating-systems

4-6 AWS CloudFormation

AWS CloudFormationは、JSONやYAML形式で記述されたテンプレートテキストファイルから、AWS上のEC2やS3などAWSリソースの環境を構築できるサービスです。AWSインフラ環境構築作業をソフトウェアコード化し、環境構築作業の迅速化、作業ミスの防止、再利用などを実現できます。

1 AWS CloudFormationのテンプレート

CloudFormationでは、定められた形式に従って記述されたテンプレートファイルを、AWSコンソール上へアップロードすることで実行します。もしくはAWS CLIを使って、テンプレートファイルを入力ファイルとしてコマンド実行することも可能です。すると、記述内容に従って、AWSリソースが自動構築されます。この構築されたリソースの集合をスタック（Stack）と呼びます。

CloudFormationでは、このスタック単位でAWSリソースの作成、変更、削除を行っていくことになります。

●テンプレートの構成要素

テンプレートは以下の10の要素で構成されます。

【CloudFormationの構成要素】

構成要素	説明	必須
Template Version	テンプレートのバージョン。最新のテンプレートの形式バージョンは2010-09-09。	
Description	テンプレートの説明を記述する。	
Metadata	Resourceに指定したAWSリソースに対する補足情報を提供する。AWSコンソール上でテンプレートを読み込み、スタックを構築した際にMetadataを使用して項目の表示順序などを制御できる。	
Parameters	AWSコンソール上でテンプレートを読み込み、スタックを構築した際に、Parametersに指定された項目はGUI上からも指定できるようになる。また、Parametersで定義した値は変数として使用できる。複数箇所で利用できるほか、テンプレートをネストした構成で親テンプレートからのプロパティ参照などにも使用される。パラメータは文字列型やNumberなど複数の型を持ち、入力値の制約なども定義可能。	

構成要素	説明	必須
Rules	Stackの作成・更新時にテンプレートに渡されるパラメータおよび組み合わせを検証する。定義したルール条件および判定のアサーションに対し、無効なパラメータが渡された場合は、Stackは作成・更新されない。	
Mappings	Mappingsでは、テンプレート内で使用する2次元キーバリューマッピングテーブルを記述できる。リージョンやユーザの入力パラメータにより、使用したいAMI（Amazonマシンイメージ）が変わってくる場合などに利用される。	
Conditions	Resourcesの記述に関して、有効となる条件を事前に定義する。Resourcesに定義するリソースにはConditionプロパティを付与でき、Conditionsで指定したパラメータが一致する場合、Resource定義の有効可否が一括で切り替わる。プロダクションやステージング環境など異なる条件でリソース定義を分けたい場合などに利用する。	
Transform	サーバレスアプリケーションの場合に使用する。AWS SAM（Serverless Application Model）のバージョンを指定する。	
Resources	Stackの構成要素となるAWSリソースを定義する。テンプレートの必須となる要素であり、AWSのリソースタイプにより定義できるプロパティは異なる。	○
Outputs	Stackを構築した後にAWSコンソール上で表示させたい項目や、ほかのテンプレートなどで取得したい情報を定義する。なお、出力できる値はリソースごとで決められており、すべてのプロパティが出力できるわけではない。	

テンプレート内のResourcesで定義するAWSリソースには、AWSが提供する主要なサービスの構築を指定できます。

●組み込み関数と擬似パラメータ

パラメータの参照や文字列操作などで利用できる組み込み関数や擬似パラメータ（Pseudo Parameters）が用意されています。関数は、「!」もしくは「Fn::」接頭辞を関数名の前に付与することで呼び出せます。擬似パラメータは\${パラメータ名}の形式で参照可能です。

主要な関数と擬似パラメータは以下のとおりです。テンプレート記載のイメージがわかりやすくなるよう表記の例も示しています。

【組み込み関数】

関数	説明	表記例
Ref	リソース論理名の物理IDを参照する。なお、Refは「Fn::」接頭辞が必要ない。	!Ref SampleCloudFormationVPC
Base64	文字列をBase64エンコードする。	!Base64 xxxx

関数	説明	表記例
Sub	変数を含む文字列を指定した変数値で置き換える。	<code>!Sub \${VPCStackName}-Output</code>
GetAtt	リソースが持つ属性値を取得する(取得可能な属性値はリソースにより異なる)。	<code>!GetAtt SampleCloudFormationVPC.CidrBlock</code>
Select	配列内の要素を指定したインデックス番号に応じて取得する。	<code>!Select [0, ["A", "B", "C"]]</code>
Join	文字列をデリミタを含めて結合する。単純な結合ならブランクを指定する。	<code>!Join ["", ["A", "B"]]</code>
ImportValue	別のスタックで使用されたリソースの出力を取り出す(クロススタックリファレンスで使用)。	<code>!ImportValue sample-cloudformation-vpc-VPCID</code>
FindInMap	Mappingsからデータを取り出す。	<code>!FindInMap [MapLogicalName, ap-northeast-1, AMI]</code>
Split	文字列を指定したデリミタで分割して配列を返却する。	<code>!Split [, "A,B"]</code>
条件関数 (If、Or、And、Not、Equals)	主にConditionsで定義した条件をもとに、Resourcesなどで条件判定の記述などに使用。	<code>!If [ConditionsLogicalName, "A", "B"]</code>

【擬似パラメータ】

サービス	用途
AWS::Region	リージョン名を取得する
AWS::StackId	スタックIDを取得する
AWS::StackName	スタック名を取得する
AWS::AccountId	AWSアカウントIDを取得する

具体的なイメージをつかむために、DynamoDBを構築する場合のサンプル(sample-dynamodb-cfn.yml)を示します。

```
AWSTemplateFormatVersion: '2010-09-09'

Description: DynamoDBを構築するサンプルテンプレート (sample-dynamodb-cfn.yml)

Parameters:
  EnvType: # パラメータ論理名
    Description: Which environments to deploy your service.
    Type: String
    AllowedValues: ["Dev", "Staging", "Production"]
```

```
Default: Dev
Conditions:
  ProductionResources: {"Fn::Equals": [{"Ref": "EnvType"}, "Production"]}
  StagingResources: !Equals [ !Ref EnvType, "Staging" ]
  DevResources: {"Fn::Equals": [{"Ref": "EnvType"}, "Dev"]}

Resources:
  DynamoDBSampleTable: # リソース論理名
    Type: AWS::DynamoDB::Table
    Properties:
      TableName: !If ["ProductionResources", "sample-table", !If ["StagingResources", "staging_sample-table", "dev_sample-table"]]
      BillingMode: PROVISIONED
      SSESpecification: !If ["ProductionResources", { "SSEEnabled": true }, !Ref "AWS::NoValue"]
    AttributeDefinitions:
      - AttributeName: samplePartitionKey
        AttributeType: S
      - AttributeName: sampleSortKey
        AttributeType: S
    KeySchema:
      - AttributeName: samplePartitionKey
        KeyType: HASH
      - AttributeName: sampleSortKey
        KeyType: RANGE
    ProvisionedThroughput:
      ReadCapacityUnits: 5
      WriteCapacityUnits: 5

Outputs:
  EnvironmentRegion: # アウトプット論理名
    Description: DynamoDBのリージョン
    Value: !Sub ${AWS::Region}
    Export:
      Name: !Sub SampleDynamoDB-${EnvType}-Region
  DynamoDBServiceEndpoint:
    Description: DynamoDBのサービスエンドポイント
    Value: !Sub https://dynamodb.${AWS::Region}.amazonaws.com
    Export:
      Name: !Sub SampleDynamoDB-${EnvType}-ServiceEndpoint
```

テンプレートの開発者は各トップ要素の下に、任意の論理名を付与します。この論理名はテンプレート内で一意となるように設定する必要があります。また、現時点では、ResourceとOutputs、Metadata、Update Policyで組み込み関数を使用できます。組み込み関数や擬似パラメータを併用したり、条件付きでスタックリソースを作成したりすることもできます。

2 CloudFormationのその他の機能

CloudFormationでは、以下のような機能をサポートしています。

●ネスティッドスタック

CloudFormationでは、複数のテンプレートでネスト（親子）を構成できます。下記のように、親したいテンプレート（sample-infra-dev-cfn.yml）で、AWS::CloudFormation::Stackタイプを使用すると、前項で示したテンプレート（sample-dynamodb-cfn.yml）を子要素として利用できます。この機能をネスティッドスタックといいます。

パラメータも親テンプレートから渡すことができますし、逆に子テンプレートのOutputs要素を用いて、親テンプレートからGetAtt関数で参照することもできます。

```
Resources:
  DynamoDBDevStack:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: ./sample-dynamodb-cfn.yml
    Parameters:
      EnvType: !Sub ${EnvType}

Outputs:
  EnvironmentRegion:
    Value: !GetAtt DynamoDBDevStack.Outputs.EnvironmentRegion
    Description: 子要素のOutputsで出力した内容を!GetAttを使用して参照する例
```

複数の環境でテンプレートを再利用したり、ネットワーク関連設定やサーバ設定など管理しやすい単位でテンプレートを分けたりしたいときに便利な機能です。

ただし、TemplateURL要素で参照している子のテンプレートは、S3にアップロードしておかなければなりません。次ページの例のとおり、AWS CLIでpackageコマンドを実行することにより、指定したS3バケットにローカルに存在するファイルをアップロードし、バケットのオブジェクトキーでTemplateURLを置き換えた

新たなテンプレートを生成できます。実際にスタックを作成する際は、新たに置き換えられたテンプレートを実行します。

```
#!/usr/bin/env bash

template_path="sample-infra-dev-cfn.yml"
output_template="sample-infra-dev-package-cfn.yml"
s3_bucket="sample-cloudformation-package"

aws cloudformation package --template-file ${template_path}
--s3-bucket ${s3_bucket} --output-template-file ${output_template}
```

●クロススタックリファレンス

スタックのOutputs要素の値は、組み込み関数「Fn::ImportValue」で参照することもできます。次のコードは、前項で示したテンプレート（sample-dynamodb-cfn.yml）でOutputs出力した内容を参照する例です。この機能をクロススタックリファレンスといい、スタックを構築したAWSアカウント内で参照できます。

```
Parameters:
  EnvType:
    Description: Which environments to deploy your service.
    Type: String
    AllowedValues: ["Dev", "Staging", "Production"]
    Default: Dev

Outputs:
  EnvironmentRegion:
    Value: Fn::ImportValue: !Sub SampleDynamoDB-${EnvType}-Region
    Description: 別のテンプレートでOutputs出力した内容をFn::ImportValueを使用して参照する例
```



ほかのテンプレートのリソースを参照する方法を問われる場合があるので、ネスティッドスタックやクロススタックリファレンスなどの方法をしっかり理解しておくようにしましょう。

●ダイナミックリファレンス

AWS Systems Manager Parameter StoreやAWS Secrets Managerなどと連携し、パスワード等の認証情報を動的に参照できます。ダイナミックリファレンスを使用するには、テンプレートに以下のように、「{{resolve:サービス名:パラメータ種別:キー名:バージョン}}」といった形式で記述します。

- ・'{{resolve:ssm:parameter-name:version}}'
- ・'{{resolve:ssm-secure:parameter-name:version}}'
- ・'{{resolve:secretsmanager:secret-id:secret-string:json-key:version-stage:version-id}}'

次のコードは、RDSでデータベースのマスターユーザのIDとパスワードをParameter Storeから取得する例です。リソースの作成にテンプレートへの直接的な記述が望ましくない秘匿データなどが必要な場合に活用するとよいでしょう。

```
Resources:
RDSInstance:
  Type: AWS::RDS::DBInstance
  UpdateReplacePolicy: Snapshot
  DeletionPolicy: Snapshot
  Properties:
    DBInstanceIdentifier: sample-rds-postgresql
    DBName: sample_database
    Engine: postgres
    MultiAZ: false
    MasterUsername: '{{resolve:ssm-secure:sample-rds-username:1}}'
    # ユーザIDを取得
    MasterUserPassword: '{{resolve:ssm-secure:sample-rds-
password:1}}' # パスワードを取得
    DBInstanceClass: db.t2.micro
    AllocatedStorage: '20'
    DBSubnetGroupName: !Ref DBSubnetGroup
    MonitoringInterval: 10
    MonitoringRoleArn: !GetAtt DBMonitorRole.Arn
    VPCSecurityGroups:
      - Fn::ImportValue: SecurityGroupRdsPostgres
```

●カスタムリソース

カスタムリソースは、CloudFormationのデフォルトのResourceとして定義されていないリソースを処理として実行する機能です。Lambda関数を実行するLambda-backedカスタムリソースと、SNS-backedカスタムリソースを実行対象として選択できます。

例えば、DynamoDB構築後に、初期データを設定する必要があるとします。これをCloudFormationで実現するには、まずDynamoDBへデータを設定するLambda関数を作成します。次に、AWS::CloudFormation::CustomResourceまたはCustom::XXXX（XXXXは任意の文字列）リソースに、作成したLambda関数のARNを指定します。これらのカスタムリソースを記述したテンプレートを実行すると、Lambda関数が実行され、DynamoDBへデータを設定する処理が実行されます。

●マクロ

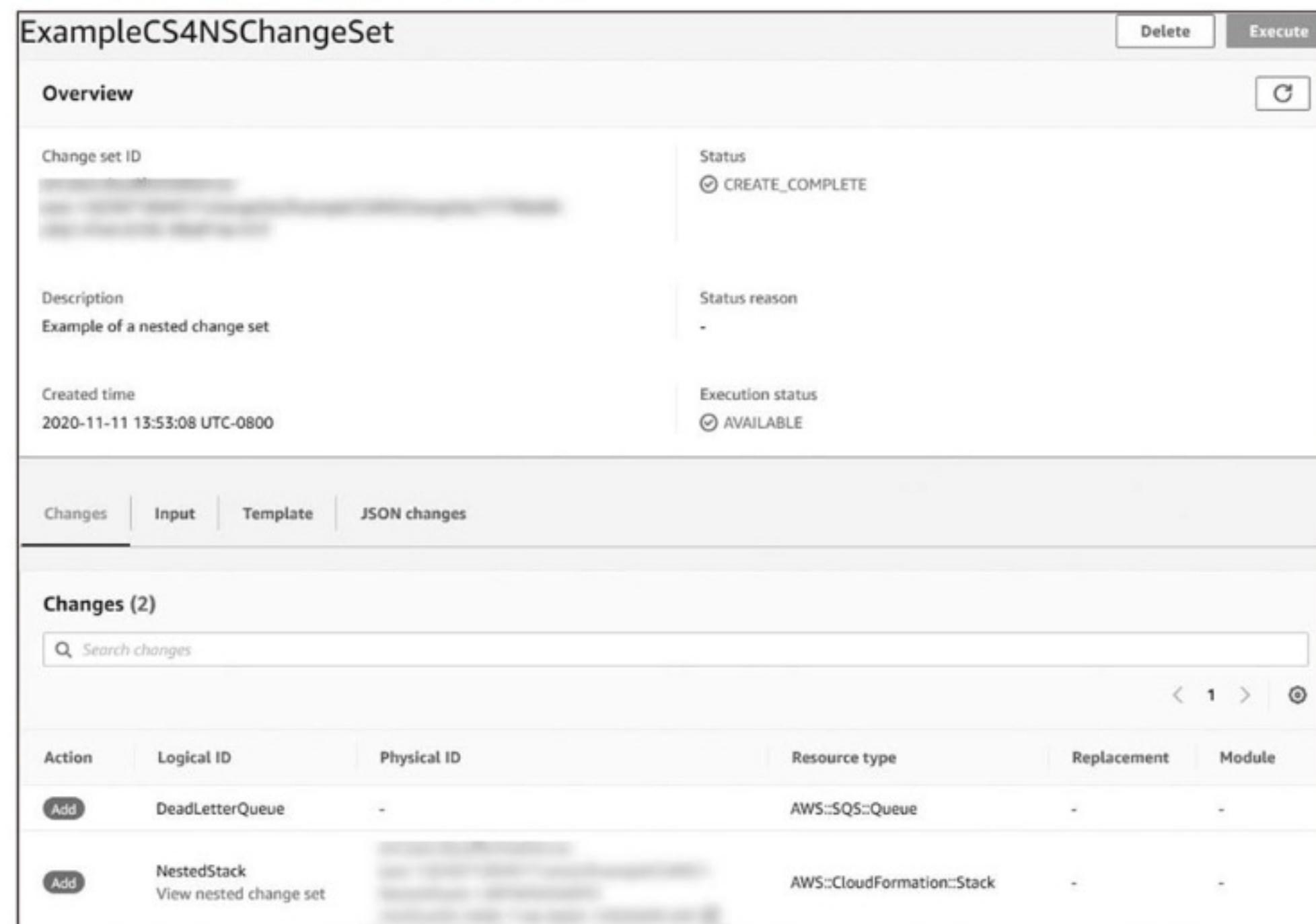
マクロはテンプレートの標準機能で実現が難しい変換処理などをLambda関数として呼び出す機能です。

例えば、たくさんのSQSのキュー定義などを作成する必要があるとします。これをCloudFormationで実現する方法を考えてみましょう。まず、SQS定義のベースになるインプットテンプレートを1つ作成し、ループ処理でSQSを複数作成・返却するLambda関数を作ります。作成したLambda関数のARNをAWS::CloudFormation::Macroリソースに指定します。このテンプレートに対し、作成するSQS定義数をインプットパラメータとして指定して実行すると、Lambda関数が実行され、指定した数だけの定義がリソースとして作成されます。

●チェンジセット

チェンジセットは、変更を要求した箇所とそれにより影響を受けるリソース論理IDを事前に確認できる機能です。プロダクション環境で動作する作成済みスタックに変更を加える際、意図しないリソースまで変更が加えられていないか確認するために活用するとよいでしょう。

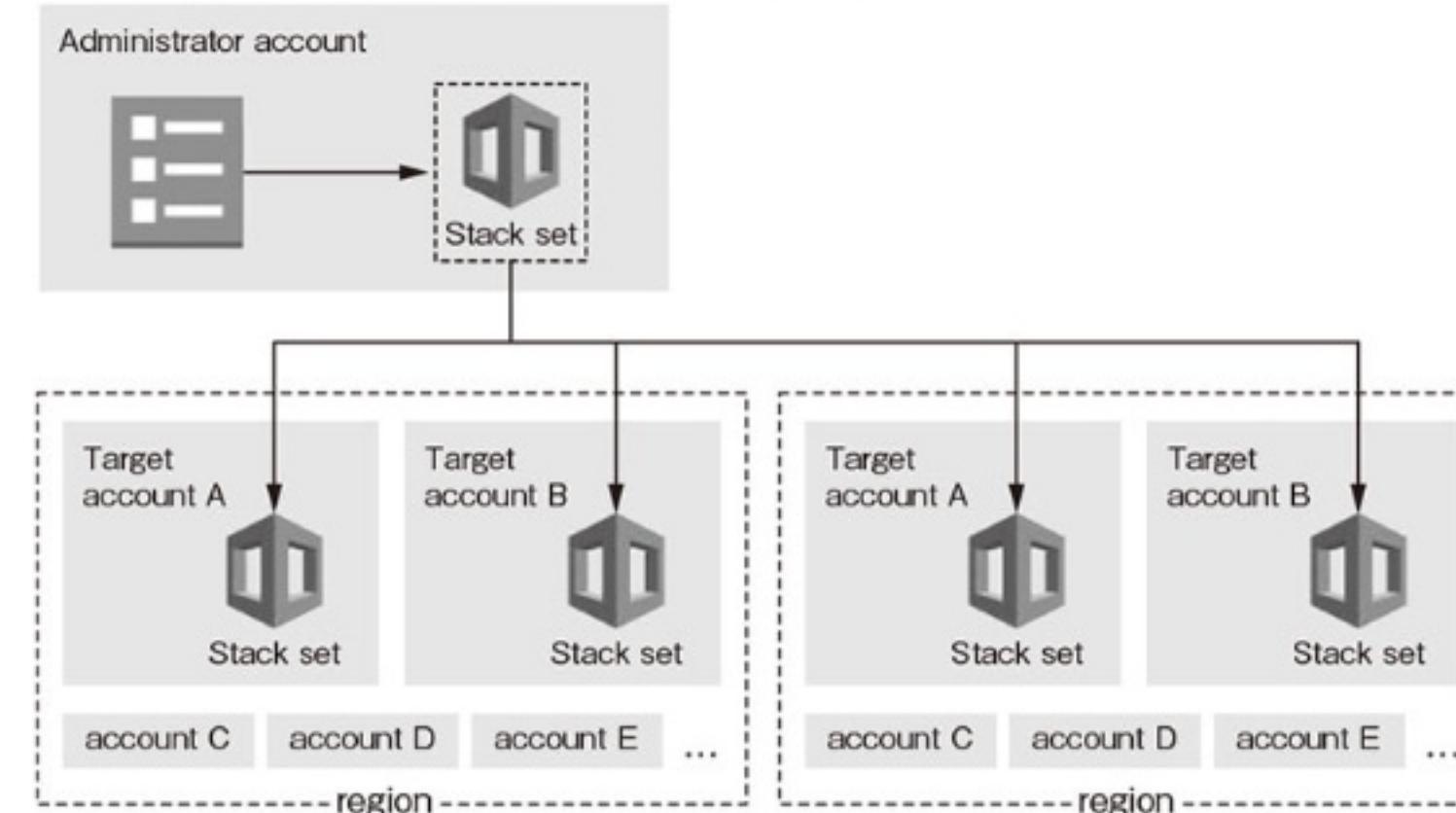
【チェンジセットのイメージ図※10】



●スタックセット

1つのテンプレートを複数のAWSアカウントおよびリージョンに展開する機能です。

【スタックセットのイメージ図※12】



複数のリージョンに同じ環境を展開する場合や、別のリージョンでテストを行う場合などに使用されます。

●ドリフト検出

ドリフト検出は、現状のリソースとテンプレート定義の差分を検出する機能です。リソース作成後に、手動で反映された更新などがあるか検出する際に有用な機能です。テンプレートと対応するスタックは複数のリソースを含みますが、スタック全体および、個々のスタックに関する差分を検出できます。

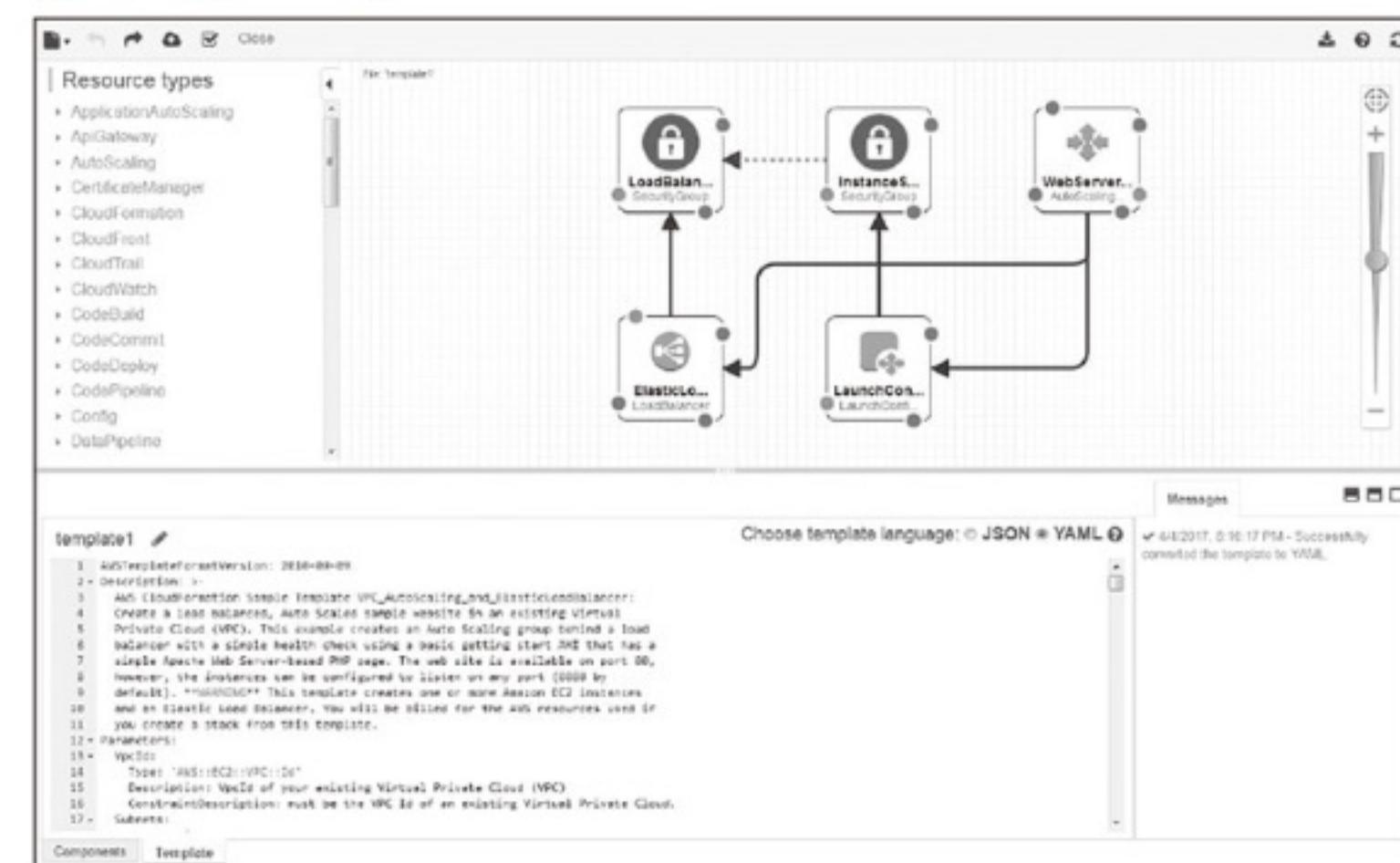
【ドリフト検出のイメージ図※11】



●CloudFormation デザイナー

AWSコンソールから利用できる、ドラッグアンドドロップでテンプレートを作成・編集できる機能です。テンプレートを作成する際のベースとして活用するとよいでしょう。

【デザイナー※13】



※10 https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets-create.html

※11 https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/detect-drift-stack.html

※12 https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/what-is-cfnstacksets.html

※13 https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/working-with-templates-cfn-designer-overview.html

●テンプレート検証・テストツール

CloudFormationでは、コード補完・文法チェックを行う統合開発環境向けプラグインやテスト機能を提供するTaskcatなど、テンプレート作成をサポートするツールが提供されています。開発を行う際は次のようなツールを組み合わせて利用するとよいでしょう。

【テンプレートの検証・テストツール】

ツール	提供元	環境	説明
validate-template コマンド	AWS	AWS CLI	AWS CLIが提供するテンプレートの検証コマンド。テンプレートの構文を確認するためのもので、リソースに対して指定したプロパティの値等が有効であることを確認するためのものではない。
IntelliJ CloudFormation プラグイン	サード パーティ	統合開発環境	IntelliJ IDEAなどの統合開発環境で動作するテンプレート検証プラグイン。IDEAのみならずJetBrains社で提供されているIDEであれば動作する。ResourceTypeとPropertiesの定義・参照検証などを実行するが、当プラグインでは、簡単な構文チェックなどは行えるものの、必須・任意パラメータの有無などの検証はできない。
cfn-python-lint	AWS	Pythonがインストールされた実行環境/エディタ	Pythonがインストールされた環境で実行可能なテンプレート検証Linter。必須・任意パラメータの有無などのテンプレートの検証機能に加え、エディタ上でコード補完・サジェスト機能がある。AtomやEmacs、NeoVim、SublimeText、IntelliJ IDEA、Visual Studioなどのエディタをサポートし、各エディタ向けにプラグインが提供されている。
Taskcat	AWS	コマンドライン	CloudFormationテンプレートをテストするオープンソースのツール。複数のリージョンでテンプレートをテスト実行できるのが特徴。テストパラメータ等も細かく設定しながら、正常/失敗終了したかどうかの証跡用レポートをHTML出力できる。

4-7 AWS Serverless Application Model

AWS Serverless Application Model (SAM) は、前節で解説したCloudFormationの拡張機能です。サーバレスアプリケーションをより簡易的な記述で定義・構築できます。

1 SAMのテンプレート

SAMはCloudFormationと同様に、JSONやYAML形式で記述されたテンプレートテキストファイルからリソースをデプロイします。AWS Lambda、Amazon API Gateway、Amazon DynamoDBなどのサーバレス関連のリソースは、CloudFormationでも構築できますが、SAMを利用するとテンプレートの記述が少なくて済み、より簡潔にリソースを定義することができます。

また、SAMでは、SAM Command Line Interface (CLI) と呼ばれるコマンドラインツールを利用して構築を行います。

●テンプレートの構成

SAMのテンプレートに記述する内容は、基本的にCloudFormationと同じです。ただし、冒頭に「Transform: 'AWS::Serverless-2016-10-31'」という宣言を記述する必要があります。この宣言を記述することで、リソースのデプロイ時に背後で行われる処理でAWS CloudFormation本来のテンプレートに変換されます。変換処理は前節で紹介したCloudFormationのマクロ機能が利用されています。AWSが用意しているマクロ用のLambda関数を使って、SAM固有の定義をCloudFormationの要素に変換するイメージです。

●Globals 要素

SAMには、CloudFormationにはない新たな要素「Globals」が追加されています。これは複数のリソースに共通する構成を一括で定義するためのものです。

Globalsで定義できる代表的な要素を次のとおり列挙します。Resourceに定義したAWSリソースは暗黙的にこのGlobals要素の設定を継承します。必要に応じて、各リソースの定義でGlobalsの設定を上書きすることも可能です。

【Globalsで指定可能な代表的な要素】

要素		説明
Functions	Runtime	Lambda関数の実行ランタイムを設定します。
	Handler	Lambda関数のハンドラを定義します。
	Environment	Lambda関数を実行する際の環境変数などを定義します。
	MemorySize	Lambda関数を実行する際のメモリサイズを定義します。
	Timeout	Lambda関数を実行する際のタイムアウト時間を定義します。
Api	Auth	API Gateway (REST API) へのアクセスを制御するための認証処理の方法（オーソライザ）を設定します。複数のオーソライザの定義が選択できます ^{*14} 。オーソライザについては4章1節API Gatewayも参照してください。
	EndpointConfiguration	REST APIのエンドポイントタイプを設定します。エンドポイントタイプについては4章1節API Gatewayも参照してください。
	Cors	API Gateway の CORS (Cross Origin Resource Sharing) を有効化するURL定義を設定します。
	Domain	API Gatewayに設定するカスタムドメインを設定します。
HttpApi	Auth	API Gateway (HTTP API) へのアクセスを制御するための認証処理の方法（オーソライザ）を設定します。複数のオーソライザの定義が選択できます ^{*15} 。オーソライザについては4章1節API Gatewayも参照してください。
	StageVariables	ステージ変数をマップ（連想配列）として定義します。

●Resource 定義

SAMでは、CloudFormationで利用できるリソースのほか、サーバレスの構築に特化した、以下のリソースタイプをResourceに定義できます。

【AWS SAM固有のリソースタイプ一覧】

Type	デプロイされるリソース
AWS::Serverless::Function	AWS LambdaのLambda関数
AWS::Serverless::LayerVersion	AWS LambdaのLambda Layers
AWS::Serverless::Api	Amazon API GatewayのREST API

*14 https://docs.aws.amazon.com/ja_jp/serverless-application-model/latest/developerguide/sam-property-api-apiauth.html

*15 https://docs.aws.amazon.com/ja_jp/serverless-application-model/latest/developerguide/sam-property-api-apiauth.html

Type	デプロイされるリソース
AWS::Serverless::HttpApi	Amazon API GatewayのHTTP API
AWS::Serverless::SimpleTable	Amazon DynamoDBのテーブル
AWS::Serverless::Application	AWS Serverless Application Repositoryの公開アプリケーション
AWS::Serverless::StateMachine	AWS Step Functionsのステートマシン

この中でぜひ覚えておきたいのが、サーバレス処理の中核となるAWS::Serverless::Functionです。このリソースタイプには実行のトリガーとなるEvents要素を定義します。利用頻度が高いのはAPI GatewayやDynamoDBなどですが、ほかにもS3のオブジェクト作成・更新・削除、SQSやKinesis、Cloud Watch Eventsが実行された場合などのイベントも選択できます。



上記の表に登場する「AWS Serverless Application Repository」はサーバレスアプリケーションのマネージドリポジトリです。自分が作成したサーバレスアプリケーションを登録して公開したり、他者が作成したサーバレスアプリケーションを検索して利用したりすることができます。公開範囲はパブリック、プライベートともに設定可能です。Lambda関数の作成時にAWS Serverless Application Repositoryで公開されているアプリケーションを選択することも可能です。

●ポリシーテンプレート

ポリシーテンプレートは、Lambda関数に権限を付与するIAMポリシー定義を簡易化したものです。CloudFormationでは長くなりがちな記述を、ポリシーテンプレートを利用することで簡易的に記述できるようになります。

サポートされている対象^{*16}のうち、代表的なものは以下のとおりです。

- AWS::SecretsManager::GetSecretValuePolicy
指定されたSecrets Managerからシークレット値を取得する権限を付与します。
- CloudFormation::DescribeStacksPolicy
CloudFormationスタックを参照する権限を付与します。
- CloudWatch::PutMetricPolicy
CloudWatchにメトリクスを送信する権限を付与します。
- CodePipeline::LambdaExecutionPolicy
Lambdaによって呼び出されたCodePipeline関数に対してジョブのステータスをレポートする権限を付与します。
- DynamoDB::CrudPolicy
DynamoDBに対する作成、読み取り、更新、および削除のアクセス許可を付与します。

*16 https://docs.aws.amazon.com/ja_jp/serverless-application-model/latest/developerguide/serverless-policy-templates.html

- EC2DescribePolicy
EC2インスタンスの詳細情報を表示する権限を付与します。
- KinesisStreamReadPolicy
Kinesis Streamを一覧表示し、読み取る権限を付与します。
- KMSDecryptPolicy
KMSキーを使用して復号する権限を付与します。
- KMSEncryptPolicy
KMSキーを使用して暗号化する権限を付与します。
- S3CrudPolicy
S3バケットオブジェクトに対する作成、読み取り、更新、および削除のアクセス許可を付与します。
- SNSCrudPolicy
SNSトピックを作成、公開、サブスクライブする権限を付与します。
- SQSSendMessagePolicy
SQSキューにメッセージを送信する権限を付与します。
- SQSPollerPolicy
SQSキューにポーリングする権限を付与します。
- SSMPParameterReadPolicy
Systems Manager Parameter Storeのパラメータ値を取得する権限を付与します。
- StepFunctionsExecutionPolicy
Step Functions ステートマシンを実行する権限を付与します。
- VPCAccessPolicy
VPCへのElastic Network Interfaceを作成、削除、デタッチするためのアクセス権を付与します。

●テンプレートのサンプル

ここまで解説してきたテンプレートの具体的なイメージをつかむために、SAMサンプルを見てみましょう。次のサンプルでは、API GatewayとLambda関数、DynamoDBテーブルをSAMで構築しています。

```
Transform: AWS::Serverless-2016-10-31 # SAM必須要素

Globals: # 共通構成を一括で定義するGlobals要素
Function:
  Timeout: 20
  MemorySize: 512
  Environment:
    Variables:
      TABLE_NAME: !Ref OrdersTable
```

```
Resources:
GetOrderFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: target/sam-java-sample.jar
    Handler: com.sample.handler.GetOrderHandler::handleRequest
    Runtime: java11
    Policies: # ポリシーテンプレート
      - DynamoDBCrudPolicy:
          TableName: !Ref OrdersTable
  Events: # Lambda実行のトリガーイベント
    GetOrder:
      Type: Api
      Properties:
        Path: /orders/{order_id}
        Method: get

OrdersTable:
  Type: AWS::Serverless::SimpleTable
```

補足ですが、SAMテンプレートにはCloudFormationで使用するリソースも定義することができます。これは、SAMテンプレートで定義した内容も、最終的にはCloudFormationの実行と同等になるためです。



SAMテンプレートで固有に定義できる要素と、どのような定義が利用できるか押さえておきましょう。

2 SAM CLI

AWS SAM CLIは、サーバレスアプリケーションの開発、SAMテンプレートの検証、デプロイなどの各種操作を実行できるコマンドラインツールです。Linux、Windows、macOSの各環境で利用できます。

サーバレスアプリケーションの開発でよく利用するSAM CLIコマンドは次のとおりです。

【AWS SAM CLIでよく利用するコマンド】

コマンド	役割
sam init	初期化処理として、SAMテンプレートやサンプルのLambda関数を生成します。
sam validate	SAMテンプレートの検証を行い、文法誤り等の問題の検出を行います。
sam build	SAMテンプレートをもとにして、アプリケーションのビルドを行います。
sam package	アプリケーションのパッケージ（ソースコードと依存関係のZIPファイル）を作成し、S3バケットにアップロードします。 AWS CLIのaws cloudformation packageコマンドと同等の機能です。
sam deploy	AWS CloudFormationへのデプロイを行います。 AWS CLIのaws cloudformation deployコマンドと同等の機能です。

上記以外にもさまざまなコマンドが用意されていますので、詳細はAWS SAMの開発者ガイドの「AWS SAM CLIコマンドリファレンス」を参照してください^{※17}。



SAMを使用した場合のデプロイの手順を押さえておきましょう。

4-8 AWS Elastic Beanstalk

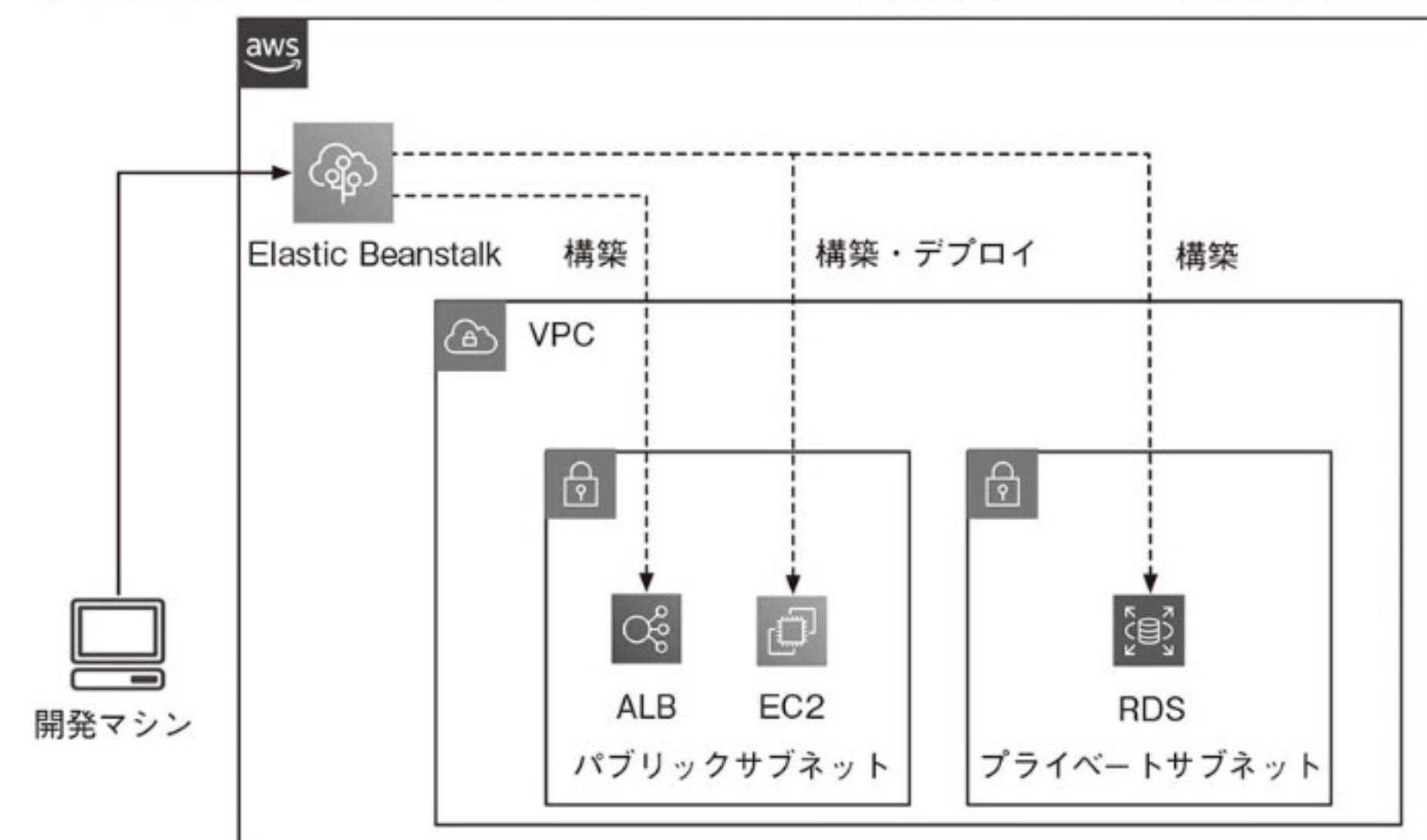
AWS Elastic Beanstalkは、典型的なシステム構成やインフラストラクチャ設定をオプションの中から選択し、自動的にアプリケーション環境を構築するサービスです。

1 Elastic Beanstalkの概要

Elastic Beanstalkは、プレゼンテーション層/アプリケーション層/データ層からなる3層Webアプリケーションや、キューを用いたバッチ処理などの典型的なプラットフォームを構築するためのサービスです。プラットフォームを選択し、アプリケーションコードをアップロードするだけで、ロードバランサー（ALB）やデータベース（RDS）をはじめとするさまざまなミドルウェアを自動で組み合わせてくれるので、インフラストラクチャを最小限の手続きで設定できます。俗にいうPaaS（Platform as a Service）を実現するサービスともいえるでしょう。

Elastic BeanstalkはAWSマネジメントコンソールのほか、EclipseやVisual Studioといったサードパーティの統合開発環境（Integrated Development Environment : IDE）のプラグインからでも利用できます。AWS CLIやSDKのほか、固有のCLI（Elastic Beanstalk CLI）やCodePipelineを経由しての操作も可能です。

【Elastic BeanstalkでWebアプリケーションの環境構築をする実行イメージ】



※17 https://docs.aws.amazon.com/ja_jp/serverless-application-model/latest/developerguide/serverless-sam-cli-command-reference.html

2 Elastic Beanstalkがサポートする構成・プラットフォームオプション

Elastic Beanstalkでは、以下のオプションを組み合わせてアプリケーション環境を構築できます。

【Elastic Beanstalkの構成・プラットフォームオプション】

項目	説明	設定項目/オプション
プラットフォーム	アプリケーションの言語や実行環境を選択します。	Docker、.NET Core on Linux、.NET on Windows Server、GlassFish、Go、Java、Node.js、PHP、Python、Tomcat、Ruby
環境	アプリケーションの種別を選択します。	・Webサーバ環境・ワーカー環境（バッチアプリケーション想定）
インスタンス構成	アプリケーションを配置するインスタンスの構成を選択できます。	<ul style="list-style-type: none"> ・冗長化構成 <ul style="list-style-type: none"> - 単一インスタンス（無料枠/スポットインスタンス） - 高可用性 - 高可用性（スポットインスタンス/オンデマンドインスタンス） ・ストレージタイプ ・メタデータ ・セキュリティグループ ・サービスロール ・キーペア ・インスタンスプロファイル
ソフトウェア設定	組み合わせて使用するソフトウェアを選択します。	<ul style="list-style-type: none"> ・AWS X-Rayの使用有無 ・Amazon S3ログストレージの使用有無 ・CloudWatch Logsの使用有無 ・環境変数の設定
オートスケーリング設定	インスタンスのオートスケーリングを設定します。	<ul style="list-style-type: none"> ・最小実行数 ・最大実行数 ・オンデマンド ・スポットの組み合わせ（フリート）の構成 ・インスタンスタイプ ・アベイラビリティゾーン ・スケーリング条件 <ul style="list-style-type: none"> - トリガーとなるメトリクスと上限・下限しきい値 - 期間

項目	説明	設定項目/オプション
ロードバランサー構成	(Webサーバ環境で高可用性構成時のみ) ロードバランサーの種類やリスナー・ルーティングルールを設定します。	<ul style="list-style-type: none"> ・ロードバランサーの種類 <ul style="list-style-type: none"> - ALB - NLB - CLB ・ポート/プロトコル設定 ・リスナールール設定 ・ルーティング設定 ・トラフィックロギング設定
デプロイポリシー設定	アプリケーションをデプロイする際のポリシーを設定します。	<ul style="list-style-type: none"> ・デプロイポリシー設定 ・デプロイバッチ（台数）サイズ/割合 ・Blue/Greenデプロイメント・URLスワップ
モニタリング設定	アプリケーションの実行インスタンスマトリクスやロードバランサーのモニタリングを設定します。	<ul style="list-style-type: none"> ・CloudWatchカスタムメトリクス ・モニタリングターゲット ・ヘルスイベントのCloudWatch Logsログストリーミング
メンテナンスウィンドウ設定	プラットフォームの自動アップデートを設定します。	<ul style="list-style-type: none"> ・更新タイミング ・更新レベル（マイナー/パッチ） ・インスタンスの置換
ネットワーク設定	Elastic Beanstalkが展開するインスタンスのネットワーク設定を行います。	<ul style="list-style-type: none"> ・VPC ・ロードバランサーのサブネット配置 ・インスタンスのサブネット配置
データベース設定	アプリケーションがアクセスするデータベースの環境設定を行います。	<ul style="list-style-type: none"> ・データベース設定 <ul style="list-style-type: none"> - データベースエンジン（mysql/oracle/postgres/sqlserver） - エンジンのバージョン - ユーザ名/パスワード ・ストレージサイズ ・スナップショットのインポート/エクスポート ・冗長化構成
ワーカー設定	(ワーカー環境選択時のみ) 主にバッチ処理を想定して、ワーカーデモンやバッチ起動の契機となるSQSキュー環境設定を行います。	<ul style="list-style-type: none"> ・キュー設定 <ul style="list-style-type: none"> - SQSキューパス - キューメッセージタイプ - 可視性タイムアウト

3 その他の留意事項

Elastic Beanstalkで押さえておきたい、そのほかの注意事項を解説します。

●環境のカスタマイズ

Elastic Beanstalkのデフォルト設定以外にないマネージドサービスを利用したり、カスタマイズを加えたりしたいときもあるでしょう。このような場合は、アプリケーションのソースコードのルートディレクトリに「.ebextensions」フォルダを作成し、設定ファイルを追加することで対応できます。サードパーティ製ソフトウェアをEC2にインストールできるほか、DynamoDBやElastiCache、SNSの設定などが可能です。

また、「環境」項目（324ページの表参照）で選択する「Webサーバ」「ワーカー」の標準構成と大きく変わるアプリケーション構成にするのであれば、CloudFormationなどを活用して環境構築を行う必要があります。

●コンテナサポート

Elastic BeanstalkでもDockerおよびDocker Composeを使用したコンテナの実行がサポートされています。単一のインスタンスまたはオートスケーリング設定されたElastic Beanstalk環境でマルチコンテナインスタンスのクラスタを起動できます。Docker Composeを使用しない場合、以下の1~3のいずれかの方法でコンテナを実行できます。

1. Dockerfileを作成して、コンソール上からアップロードする。
2. コンテナイメージをビルドして、リポジトリへプッシュする。以下のような Dockerrun.aws.jsonという名称のJSON形式のファイルを作成し、コンソール上へアップロードする。JSONファイルにはイメージ名のほか、実行ポート、ターゲットボリュームなどといったコンテナ実行に必要な情報を指定する。

```
{
  "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"
}
```

3. Zip形式のアプリケーションアーカイブファイルを作成し、コンソール上からアップロードする。アーカイブファイル内のソースコードルートディレクトリには、DockerfileおよびDockerrun.aws.jsonを配置する必要がある。

4-9 その他の展開関連サービス

AWSでは、DevOpsや基盤自動化を実現するためのサービスがあり、認定試験でも内容を問われる場合があります。以下で説明するサービスは押さえておくようにしましょう。

1 Amazon OpsWorks

「Amazon OpsWorks」は、オープンソースの基盤自動化ツールであるPuppetやChefを使用するマネージドサービスです。OpsWorksでは、PuppetやChefの2種類のオープンソースに加えて、以下3種類のマネージドサービスが提供されています。

【OpsWorks】

サービス	説明
AWS OpsWorks for Puppet Enterprise	Puppetマスターサーバのマネージドサービス。サーバのバックアップやEC2インスタンス上でクライアントとなるノードの設定・デプロイ・管理を自動化する。
AWS OpsWorks for Chef Automate	Chefサーバのマネージドサービス。Chefが提供する継続的インテグレーションを実現するための「Chef Automate」や「Chef Infra」、「Chef InSpec」をインストール・管理したEC2が提供される。
AWS OpsWorks スタック	クラウドリソースの構築をChefレシピ（テンプレートに相当）を使用して行うオリジナルのマネージドサービス。Chefレシピを使用するが、Chefサーバは不要。ヘルスチェックや自動回復機能を持つ独自の管理サーバによりリソースの構築を行う。

いずれのサービスも、アクションを記録するCloudTrailと統合されており、作業証跡や監査を残す場合の負荷を軽減できます。また、管理サーバのバックアップや冗長化などAWSの特徴であるマネージドサービスの恩恵を受けられます。PuppetやChefを利用する場合は積極的に活用したいサービスです。

2 AWS AppConfig

AWS AppConfigはSystems Managerの機能の1つであり、環境変数やデータベース接続情報など環境依存のデータ定義ファイルなどを管理するサービスです。設定値の検証やデプロイモニタリングなど安全なアプリケーション環境の構築に寄与します。

3 AWS CodeArtifact

AWSが提供する、さまざまなアプリケーション・ソフトウェアパッケージライブラリのためのリポジトリサービスです。インターネット上で公開されているオープンなライブラリを中間的に保存したりするアーティファクトリポジトリとして機能します。

例えば、Java言語での代表的なアプリケーションビルドツールであるMavenが参照する、Maven Centralが提供するオープンソースのライブラリモジュール群をホストし、クローズドな環境での利用を可能にします^{*18}。

4 AWS Copilot

AWS CopilotはコンテナサービスであるAmazon ECSでコンテナ実行をする際に利用できるコマンドラインインターフェースです。プロダクションレディな環境セットを少しのコマンドでデプロイできるようにデザインされています。

ECSコンテナを用いたアプリケーション展開をCloudFormationなどと同様、迅速に展開できるサービスです。コマンドラインベースのスクリプトによる環境構築を実現する際に手軽に扱えるツールなので、サードパーティサービスによる基盤自動化を実現する際の選択肢の1つとして検討してみるとよいでしょう。

*18 <https://aws.amazon.com/jp/blogs/news/software-package-management-with-aws-codeartifact/>

演習問題

1 ある開発プロジェクトがスタートし、AWS CodeCommitで成果物の管理を行うため、環境を構築しています。開発環境にはセキュリティ対策として、インターネット通信を行う場合、HTTP/HTTPSプロキシサーバを経由しなければならない制約があります。AWS CodeCommitに接続するために端末に必要な環境設定で正しいものを選択してください。

- A. 公開鍵と秘密鍵のペアを作成し、公開鍵をAWSコンソール上から開発者のIAMユーザに設定し、秘密鍵をローカル端末に設定します。
- B. 公開鍵と秘密鍵のペアを作成し、秘密鍵をAWSコンソール上から開発者のIAMユーザに設定し、公開鍵をローカル端末に設定します。
- C. IAMユーザでGit認証情報を払い出し、開発端末のツール等に設定します。
- D. IAMポリシーにCodeCommitへのアクセス許可を追加し、開発者のIAMユーザに設定する。開発者にはクレデンシャルを作成し、端末にAWS認証情報として保存します。

2 現在稼働中のアプリケーションの資材はGitHubリポジトリ上に保存されています。今回、組織内でリポジトリを一元化する施策が推進され、AWS CodeCommitに移管することになりました。移行はGitHubから複製されたローカル端末にある最新のリポジトリを、CodeCommitへプッシュする方法を採用します。必要な環境設定や手順のうち、正しいものを選択してください。

- A. GitHubの認証トークンを払い出し、ローカル端末に設定します。
- B. IAMユーザのクレデンシャルを作成して、端末にAWS認証情報として保存します。
- C. 公開鍵と秘密鍵のペアを作成し、公開鍵をAWSコンソール上から開発者のIAMユーザに設定し、秘密鍵をローカル端末に設定します。
- D. 公開鍵と秘密鍵のペアを作成し、秘密鍵をAWSコンソール上から開発者のIAMユーザに設定し、公開鍵をローカル端末に設定します。

3 ある組織では別の部署が開発したソースコード資材を一部流用して新たな開発プロジェクトを進めようとしています。新しいプロジェクトでは、新たにAWSアカウントが払い出され、別の部署が開発したソースコードが保存されているAWS CodeCommitにアクセスしようとしています。必要な環境設定で誤っているものを2つ選択してください。

- A. 新しいプロジェクトの開発用端末にgit-remote-codecommitツールをインストールします。
- B. 流用するソースコードを保持する部署のアカウント内にクロスアカウントアクセス用のIAMロールを作成し、新しいプロジェクトのアカウントを信頼されたエンティティとして指定します。
- C. 新しいプロジェクトのアカウント内にクロスアカウントアクセス用のIAMロールを作成し、流用するソースコードを保持する部署のアカウントを信頼されたエンティティとして指定します。
- D. 新しいプロジェクトの開発用端末にAWS CLIをインストールして、プロファイルなどアクセスに必要な情報をセットアップします。
- E. 流用するソースコードを保持する部署でCodeCommitへのアクセスに利用されている公開鍵/秘密鍵ペアのうち、公開鍵を受け取り、新しいプロジェクトの開発用端末にセットアップします。

4 ある組織ではすでに商用リリースしたアプリケーションの機能を追加開発しています。追加開発している機能は、稼働中のアプリケーションのソースコードが保存されているMainブランチをベースに新たにFeatureブランチとして作成したものです。しかし、本番中に障害が発生し、Hotfixブランチにて修正対処を行い、その後、Mainブランチにその修正が取り込まれています。同じ不具合を発生させないために、Featureブランチに対して、追加機能の開発者が実行するべき操作はどれか選択してください。

- A. Mainブランチからgit fetchを行い、Mainブランチの内容をgit mergeします。必要に応じてコンフリクトを修正します。
- B. Mainブランチに対しAWS CLIでCreatePullRequestを実行し、Mainブランチの内容をgit mergeします。必要に応じてコンフリクトを修正します。
- C. Mainブランチに対しAWS CLIでCreatePullRequestを実行し、Mainブランチの内容をgit fetchします。必要に応じてコンフリクトを修正します。
- D. Mainブランチに追加開発の影響が出ないよう、Hotfixブランチに対しAWS CLIでCreatePullRequestを実行し、Hotfixブランチの内容をgit mergeします。必要に応じてコンフリクトを修正します。

5 現在稼働中のアプリケーションで内部の犯行者による不正なアクセスが発生し、この原因になったソースコードに加えられた変更履歴を調査しています。開発チームがとるべき再発防止策の中で実現できないものを選択してください。

- A. CloudTrailを有効化して、CodeCommitに対して実行されたAPIコードやGitコマンドのイベントを記録し、ログを保存します。
- B. CodeGuru Reviewerを使用して問題となる不正アクセスの可能性があるソースコードを分析し、コードレビューを行います。
- C. 商用環境にリリースされるブランチに対する変更のプルリクエストのイベントに対し、コードオーナーへ通知を行い承認を得るようにします。
- D. CloudWatch Eventでリポジトリをモニタリングし、不正アクセスの可能性があるソースコードがあった場合、開発者に通知します。

6 ある開発プロジェクトがスタートし、AWS CodeBuildを使って継続的インテグレーションを実現するための環境を検討しています。CodeBuildで実現可能なもののうち、誤っているものを選択してください。

- A. 多数の開発者が同時にリポジトリにプッシュやコミット、プルリクエストを行うため、無影響確認テストを多数同時に行えます。
- B. ビルド処理はデフォルトのコンテナイメージであるAmazon Linux 2とUbuntuに加えて、任意のカスタムコンテナイメージも利用できます。
- C. デフォルトのコンテナイメージには、さまざまなプログラミング言語のランタイムやGitやドライバなどが標準でインストールされており、個別にインストールする必要はありません。
- D. 任意の時間にビルドジョブをイベント実行するにはCloudWatch Eventsと連携する必要があります。

7 ある開発プロジェクトがスタートし、AWS CodeBuildを使って継続的インテグレーションを実現するための環境を構築しています。コンテナイメージをアウトプットする最初のビルドジョブの実行確認でジョブが異常終了しました。考えられる原因とその対処のうち、誤っているものを選択してください。

- A. コンテナイメージを生成するビルドジョブのコンテナに割り当てられたスペックが小さすぎて処理が完了せずタイムアウトを起こしたため、コンテナのメモリだけを大きく設定しました。
- B. コンテナイメージを生成するビルドジョブの実行環境として割り当てたVPCのサブネットにNAT Gatewayが設定されておらずタイムアウトを起こしたため、ルートテーブルの設定を見直しました。
- C. コンテナイメージを生成するビルドジョブの実行環境として割り当てたVPCのサブネットにアウトバウンド接続が許可されていないため、セキュリティグループの設定を見直しました。
- D. コンテナイメージを生成するビルドジョブに権限が足りていなかったため、実行モードを見直しました。

8 ある開発者がCodeBuildを使ってビルドジョブを実行するためのbuildspec.ymlを記述しています。開発者が留意すべきbuildspec.ymlおよびビルドジョブの仕様で正しいものを選択してください。

- A. buildspec.ymlは通常ソースコードルートディレクトリに配置する必要がありますが、ビルドジョブ単位に実行するファイルを指定できるため、単体試験やステージング環境向けの結合テストジョブなど複数の用途で作成しておくことができます。
- B. buildspec.ymlにはSystems Manager Parameter Storeの値を参照する際、バージョンを指定することができます。
- C. ビルドジョブでデフォルトで新規に作成されるサービスロールにSystems Manager Parameter Storeへのアクセス権限が含まれています。
- D. 1つのビルドプロジェクト内で、複数のジョブを同時実行することはできません。

9 ある開発者がCodeBuildを使ってビルドジョブを実行するためのbuildspec.ymlを記述しています。開発者が留意すべきbuildspec.ymlおよびビルドジョブの仕様で正しいものを選択してください。

- A. ビルド実行中に障害が発生しても、各フェーズ内で、on-failure属性をCONTINUEに指定しておけば、複数のリストで記述されたコマンドを続けて続行できます。
- B. runtime-versions属性で指定可能なものはコンテナのOSイメージであり、各プログラミング言語のバージョンまで指定することはできません。pre_buildフェーズで使用するランタイムを指定します。
- C. ビルド実行中に障害が発生しても、finally要素に記述されたコマンドは必ず実行されます。
- D. アーティファクトとして指定可能なファイルは原則1つだけです。

10 ある開発者がCodeBuildのビルドジョブを検証しようとしています。開発者がCodeBuild Localを使う場合の誤った記述を選択してください。

- A. CodeBuild Localを実行するためには、DockerおよびBashスクリプトが実行できる環境が必要です。
- B. CodeBuild Localを実行するためには、ビルド用のコンテナイマージが必要です。
- C. CodeBuild Localを実行するためには、CodeBuildエージェントが含まれるコンテナイマージが必要です。
- D. CodeBuild Localを実行するためには、Privilegedモードで起動しておく必要があります。

11 ある開発者がCodePipelineを使用して、ステージング環境でテストを実行したのちに、プロダクション環境へリリースするパイプラインを構築することを検討しています。選択肢の中で実現できないものを選択してください。

- A. デプロイステージのターゲットとしてAmazon ECSを選択し、コンテナリポジトリとしてDocker Hubを使用することができます。
- B. ソースステージのターゲットとして、ソースコードリポジトリとしてDocker Hubを選択することができます。
- C. テストステージのターゲットとして、テストジョブを実行するためにJenkinsを選択することができます。
- D. 任意のステージで、ワークフローを定義するためにStep Functionsを選択することができます。

12 あるエンターテインメント事業を手掛ける企業から、新しいビジネスを行うためのアプリケーション開発を受託しました。このアプリケーションのリリースには委託元企業の責任者の承認を受けたのちに実行する必要があります。CodePipelineを用いてこの承認プロセスを自動化するために必要な設定で最も簡易なものを選択してください。

- A. パイプラインでInvokeステージを作成し、レビュー用のURLや承認後にパイプラインが自動起動するリンクをAWS Lambdaを用いて作成し、責任者にメール通知します。
- B. パイプラインでInvokeステージを作成し、Amazon SNSを用いて、責任者にメール通知します。
- C. パイプラインでApprovalステージを作成し、レビュー用のURLや承認後にパイプラインが自動起動するリンクをAWS Lambdaを用いて作成し、責任者にメール通知します。
- D. パイプラインでApprovalステージを作成し、Amazon SNSを用いて、責任者にメール通知します。

13 ある開発プロジェクトではソースコード管理にGitHubを使用しています。プロダクション環境へデプロイするためのプロセスをCodePipelineで構築することになりました。必要な環境設定で誤っているものを2つ選択してください。

- A. パイプラインのSourceステージでGitHubを選択し、OAuth 2.0のWeb IDフェデレーションを使用して、GitHubに接続し、対象のリポジトリとブランチを選択します。
- B. GitHubで個人アクセストークンを作成します。パイプラインのSourceステージでGitHubを選択し、トークンを使用してGitHubへ接続し、対象のリポジトリとブランチを選択します。
- C. GitHubにログインし、対象のリポジトリとブランチのWebフック条件を設定します。
- D. Sourceステージの設定で、GitHubにある対象のリポジトリとブランチのWebフック条件を設定します。
- E. Sourceステージの設定で、GitHubにある対象のリポジトリとブランチをCodePipelineからポーリングする設定を行います。

14 新しくリリースするアプリケーションに対して、プロダクション環境と同等のステージング環境で自動テストするプロセスを開発チームが検討しています。ステージング環境はALB、ECS、RDSを用いたバックエンドサービスアプリケーションで構成されます。パイプラインのソースステージでCodeCommitからソースコードをチェックアウトします。次のビルドステージでCodeBuildを使ってアプリケーションをビルドし、アプリケーションのコンテナイメージを作成してコンテナレジストリにプッシュします。デプロイステージでECSにコンテナをデプロイします。テストステージでCodeBuildを使って、テストコードをチェックアウトし、自動テストを実行します。テストで問題ないことを確認したら、プロダクション環境へ最終的にリリースします。このプロセスの中で実現可能な、最も費用対効果の高い手法を選択してください。

- A. CloudFormationを使って、パイプラインの最初のステージでステージング環境を構築します。
- B. テストステージで、自動化テストを実行するために必要な最新のツールをインストールし、テストコードを逐次実行します。
- C. 自動テスト後に迅速にリリースするため、CodeBuildでプロダクション環境向けのコンテナアプリケーションをテスト後にビルドして、レジストリにプッシュしておきます。
- D. 次回のリリースに備えて、迅速にテストを行うために、ステージング環境を残しておきます。

15 ある開発プロジェクトはセキュリティ要件が非常に厳しく、インターネットに接続しない閉塞的なAWSネットワーク環境下でCI/CDパイプラインを構築する必要があります。このユースケースに適合する環境設定で必要な構成を選択してください。

- A. CodePipeline、S3などのAWSリージョンサービスへ接続するためのVPCエンドポイントを設定します。
- B. CodeBuildがVPCへ接続するためのプライベートサブネットを設定します。
- C. ソースコードを管理するためのオープンソースGitリポジトリをVPC内に構築します。
- D. パイプラインの各ステージでビルド・テストジョブを実行するためには必要なオープンソースのコンテナレジストリをVPC内に構築します。

16 ある開発者がCodeDeployを用いたアプリケーションのデプロイメント戦略を検討しています。EC2で利用できるデプロイ戦略を2つ選択してください。

- A. Blue/Greenデプロイメント戦略
- B. Canaryデプロイメント戦略
- C. Linearデプロイメント戦略
- D. In-Placeデプロイメント戦略
- E. Rolling-additional-batchデプロイメント戦略

17 あるアプリケーションはCodeDeployを使ってECS上にデプロイされています。現在はすべてのトラフィックを新たにデプロイしたコンテナへすべてシフトしていますが、前回リリース時に障害が発生してサービスが停止したため、今回のリリースからリクエストの20%程度を新しいアプリケーションがリリースされたコンテナへシフトさせ、30分程度状況を見て、問題なければ残りのアプリケーションを更新したいと考えています。開発チームが実施しなければならないデプロイ設定で正しいものを選択してください。

- A. デフォルトで用意されているLinearタイプの20ステップ・30分間隔のデプロイ設定をデプロイグループにアタッチします。
- B. デフォルトで用意されているCanaryタイプの20ステップ・30分間隔のデプロイ設定をデプロイグループにアタッチします。
- C. カスタムでLinearタイプの20ステップ・30分間隔のデプロイ設定を作成し、デプロイグループにアタッチします。
- D. カスタムでCanaryタイプの20ステップ・30分間隔のデプロイ設定を作成し、デプロイグループにアタッチします。

- 18** 新たにプロジェクトに参画した新人の開発者が以下の `appspec.yml` を実装しており、あなたは内容をレビューしています。指摘として正しいコメントを選択してください。

```
version: 1.0
resources:
- TargetService:
  Type: AWS::ECS::Service
  Properties:
    TaskDefinition: "arn:aws:ecs:ap-northeast1:XXXXXXXXXX:task-definition/sample-app-task-definition:1"
    LoadBalancerInfo:
      ContainerName: SampleAppContainer
      ContainerPort: 8080
Hooks:
- ApplicationStop: "ApplicationStopHookLambdaFunction"
- BeforeInstall: "BeforeInstallHookLambdaFunction"
```

- A. 必須要素である「os」要素が含まれていません。実行時にエラーとなります。
- B. AppSpec ファイルの最新バージョンは 2.0 で、できる限り最新バージョンで実行する必要があります。
- C. フック設定の ApplicationStop は EC2 やオンプレミスサーバのみでサポートされている要素です。
- D. これは ECS サービスのデプロイ配置仕様を実装しているものであり、Lambda のデプロイのフック設定が誤って記述されています。

- 19** AWS クラウドで運用しているアプリケーションと同じく、既存のオンプレミス環境でデプロイ方法を統一しようと、開発チームが CodeDeploy の導入を検討しています。オンプレミス環境の管理者から CodeDeploy の導入に必要な要件をヒアリングされました。提示する以下の要件で正しいものを選択してください。

- A. エージェントのインストールおよび動作がテストされているオペレーティングシステムは Linux サーバであり、Windows サーバはサポート対象外です。
- B. CodeDeploy エージェントはポート 443 番を使用したアウトバウンド通信が必要です。デプロイ対象となるサーバに対するファイアウォールでこの通信を許可しなければなりません。
- C. CodeDeploy エージェントの実行には Java 8 のインストールが前提となります。
- D. エージェントがリクエストを送信するために、適切なアクセス権限が必要なため、エージェントがインストールされるサーバでユーザを作成し、ロールを割り当てなければなりません。

- 20** ある開発プロジェクトはセキュリティ要件が非常に厳しく、インターネットに接続しない閉塞的な AWS ネットワーク環境下で CI/CD パイプラインを構築する必要があります。デプロイ用途で CodeDeploy を使用する場合の正しい設定を選択してください。

- A. Lambda のデプロイでは、`codedeploy` および、`codedeploy-commands-secure` の 2 種類の VPC エンドポイントを設定する必要があります。
- B. EC2 のデプロイでは、`codedeploy` および、`codedeploy-commands-secure` の 2 種類の VPC エンドポイントを設定する必要があります。
- C. ECS のデプロイでは、`codedeploy` および、`codedeploy-commands-secure` の 2 種類の VPC エンドポイントを設定する必要があります。
- D. Lambda のデプロイでは、すべて VPC 外で完結するため、VPC エンドポイントを設定する必要はありません。

- 21** 新たにプロジェクトに参画した新人の開発者が以下のCloudFormationテンプレートを実装しており、あなたは内容をレビューしています。指摘として正しいコメントを2つ選択してください。

```

Parameters:
  EnvType:
    Type: String
    AllowedValues: [ "Dev", "Staging", "Production" ]
    Default: Dev

Conditions:
  ProductionResources: { "Fn::Equals" : [{"Ref": "EnvType"}, "Production"] }
  StagingResources: !Equals [ !Ref EnvType, "Staging" ]
  DevResources: { "Fn::Equals" : [{"Ref": "EnvType"}, "Dev"] }

Resources:
  DynamoDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      TableName: !If [ "ProductionResources", "order-table", !If
        ["StagingResources", "staging_order-table", "dev_order-table"] ]
      BillingMode: PROVISIONED
      SSESpecification: !If [ "ProductionResources", { "SSEEnabled" :
        true }, !Ref "AWS::NoValue" ]
      AttributeDefinitions:
        - AttributeName: orderId
          AttributeType: S
      KeySchema:
        - AttributeName: orderId
          KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5

  DynamoDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      TableName: !If [ "ProductionResources", "customer-table", !If
        ["StagingResources", "staging_customer-table", "dev_customer-table"] ]
      BillingMode: PROVISIONED
      SSESpecification: !If [ "ProductionResources", { "SSEEnabled" :
        true }, !Ref "AWS::NoValue" ]
      AttributeDefinitions:
        - AttributeName: customerId
          AttributeType: S

```

```

KeySchema:
  - AttributeName: customerId
    KeyType: HASH
ProvisionedThroughput:
  ReadCapacityUnits: 5
  WriteCapacityUnits: 5

```

- A. 必須要素である「AWSTemplateFormatVersion」要素が含まれていません。実行時にエラーとなります。
- B. Resourcesに設定する論理名は一意である必要があるため、このテンプレートを実行するとエラーになります。
- C. 組み込み関数内で組み込み関数は利用できないため、このテンプレートは実行するとエラーになります。
- D. 組み込み関数内で擬似パラメータは利用できないため、このテンプレートは実行するとエラーになります。
- E. クロススタックリファレンスが実行できないため、Outputsで必要なパラメータを出力するべきです。

- 22** ある開発プロジェクトではAWS上で稼働するアプリケーション基盤を作成するためにCloudFormationを使用することを検討しています。環境は開発、ステージング、プロダクションなど複数あるため、テンプレートを複数の環境で再利用できるように構成したいと考えています。次の選択肢の中でCloudFormationの機能で実現できないものを選択してください。

- A. Conditions要素で指定したパラメータを使って、Resourcesに定義したリソースの各プロパティでIFなどの条件関数を使って環境ごとに設定するパラメータを切り替えます。
- B. Resourcesに定義するリソースにはConditionプロパティがあり、Conditionsで設定した条件と一致した場合にのみ、リソース作成を行なうオプションを選択できます。
- C. ネスティッドスタックで子のテンプレートに渡すパラメータを切り替えることができるため、環境ごとに依存値を定義した親テンプレートと、環境に依存しない子テンプレートを作ることで柔軟に対応できます。
- D. スタックセットを利用することにより複数のリージョンに同じ環境を展開できます。同一アカウントで重複が許されないパラメータもスタックセットの機能でLambda関数を使用し、動的に切り替えることが可能ですが、CloudFormationの機能で実現できません。