Hewlett Packard Enterprise

HPE Codar

ソフトウェアバージョン: 1.60

コンセプトガイド

ドキュメントリリース日 : 2016年 1月 (英語版)
ソフトウェアリリース日 : 2016年 1月

ご注意

保証

Hewlett Packard Enterprise Development LP製品、またはサービスの保証は、当該製品、およびサービスに付随する明示的な保証文によってのみ規定されるものとします。ここでの記載は、追加保証を提供するものではありません。ここに含まれる技術的、編集上の誤り、または欠如について、HPEはいかなる責任も負いません。

ここに記載する情報は、予告なしに変更されることがあります。

権利の制限

機密性のあるコンピューターソフトウェアです。これらを所有、使用、または複製するには、HPEからの有効な使用許諾が必要です。商用コンピューターソフトウェア、コンピューターソフトウェアに関する文書類、および商用アイテムの技術データは、FAR12.211および12.212の規定に従い、ベンダーの標準商用ライセンスに基づいて米国政府に使用許諾が付与されます。

著作権について

商標について

Adobe™は、Adobe Systems Incorporated (アドビシステムズ社) の登録商標です。

Microsoft®およびWindows®は、米国におけるMicrosoft Corporationの登録商標です。

UNIX®は、The Open Groupの登録商標です。

本製品には、'zlib' (汎用圧縮ライブラリ) のインタフェースが含まれています。'zlib': Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

ドキュメントの更新情報

このマニュアルの表紙には、以下の識別情報が記載されています。

- ソフトウェアバージョンの番号は、ソフトウェアのバージョンを示します。
- ドキュメントリリース日は、ドキュメントが更新されるたびに変更されます。
- ソフトウェアリリース日は、このバージョンのソフトウェアのリリース期日を表します。

更新状況、およびご使用のドキュメントが最新版かどうかは、次のサイトで確認できます。
https://softwaresupport.hp.com

このサイトを利用するには、HPE Passportへの登録とサインインが必要です。HPE Passport IDの登録は、次のWebサイトから行なうことができます。**https://hpp12.passport.hp.com/hppcf/createuser.do**

または、ソフトウェアサポートページの [**登録**] リンクをクリックします。

適切な製品サポートサービスをお申し込みいただいたお客様は、更新版または最新版をご入手いただけます。詳細は、HPEの営業担当にお問い合わせください。

サポート

HPEソフトウェアサポートオンラインWebサイトを参照してください。**https://softwaresupport.hp.com**

このサイトでは、HPEのお客様窓口のほか、HPEソフトウェアが提供する製品、サービス、およびサポートに関する詳細情報をご覧いただけます。

HPEソフトウェアオンラインではセルフソルブ機能を提供しています。お客様のビジネスを管理するのに必要な対話型の技術サポートツールに、素早く効率的にアクセスできます。HPソフトウェアサポートのWebサイトでは、次のようなことができます。

- 関心のあるナレッジドキュメントの検索
- サポートケースの登録とエンハンスメント要求のトラッキング
- ソフトウェアパッチのダウンロード
- サポート契約の管理
- サポート窓口の検索
- 利用可能なサービスに関する情報の閲覧
- 他のソフトウェアカスタマーとの意見交換
- ソフトウェアトレーニングの検索と登録

一部のサポートを除き、サポートのご利用には、Passportユーザーとしてご登録の上、サインインしていただく必要があります。また、多くのサポートのご利用には、サポート契約が必要です。Passport IDを登録するには、次のWebサイトにアクセスしてください。

https://hpp12.passport.hp.com/hppcf/createuser.do

アクセスレベルの詳細については、次のWebサイトをご覧ください。

https://softwaresupport.hp.com/web/softwaresupport/access-levels

HPE Software Solutions Nowは、HPSWのソリューションと統合に関するポータルWebサイトです。このサイトでは、お客様のビジネスニーズを満たすHPE製品ソリューションを検索したり、HPE製品間の統合に関する詳細なリストやITILプロセスのリストを閲覧することができます。このサイトのURLは**http://h20230.www2.hp.com/sc/solutions/index.jsp**です。

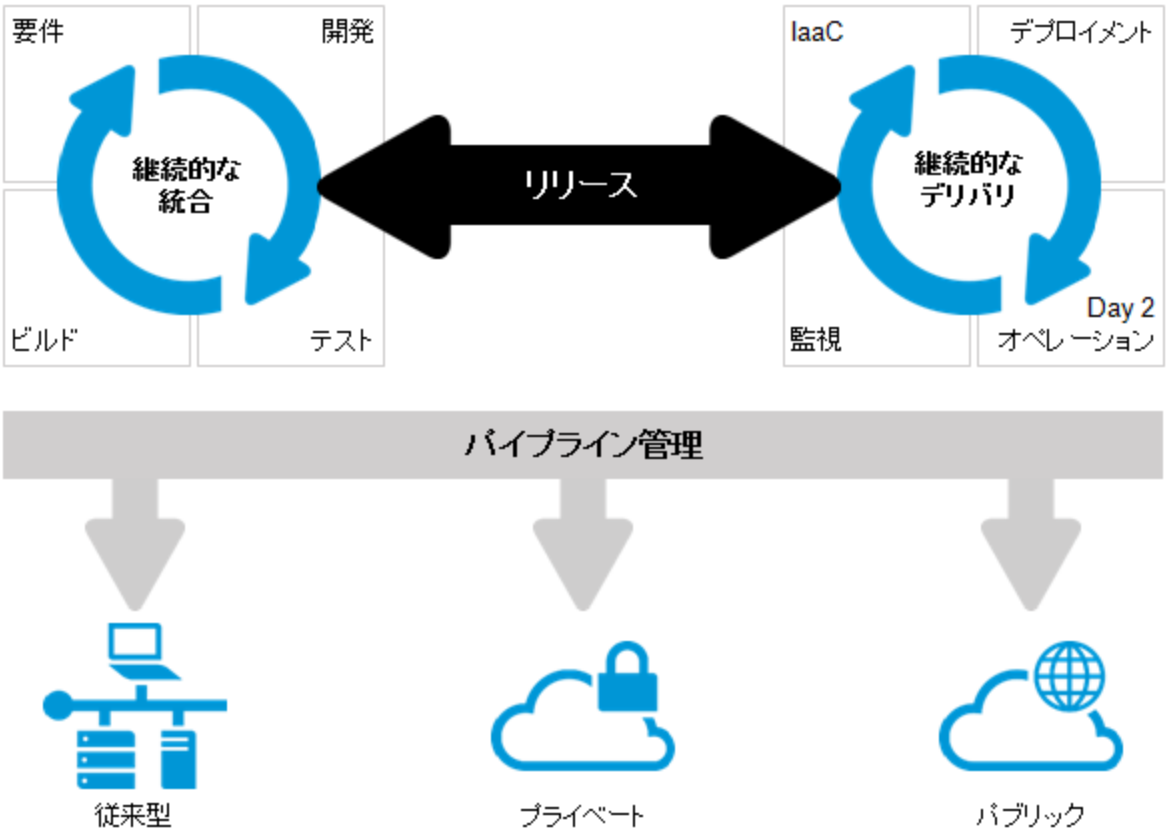
目次

Codar

組織が継続的統合を継続的デリバリに拡張する際は、新しい課題に直面します。その課題には、開発環境から運用環境に、それぞれの環境の違いを考慮しながら一貫性を持ってアプリケーションをデプロイすることがあります。

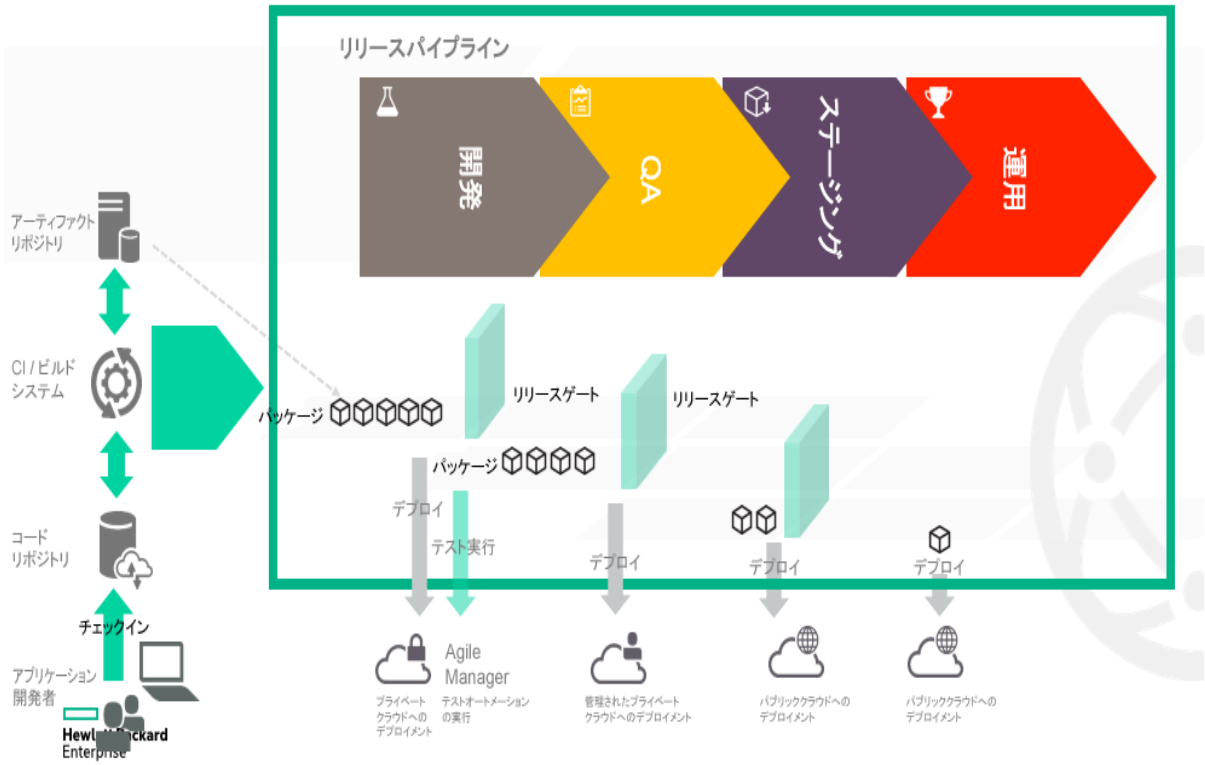
DevOpsは、コラボレーション、自動化、ガバナンスに関する一連の方針、方法、手法を使用して、開発(Dev)環境と運用(Ops)環境の溝を埋めるためのフレームワークを提供します。この目的は、ビルドまたはアセンブリの継続的な統合を、異種環境間で、再現性があり一貫性のあるアプリケーションデプロイメントに広げることです。

次の図は、DevOps環境での継続的統合と継続的デリバリのサイクルを示しています。



Codarの概要

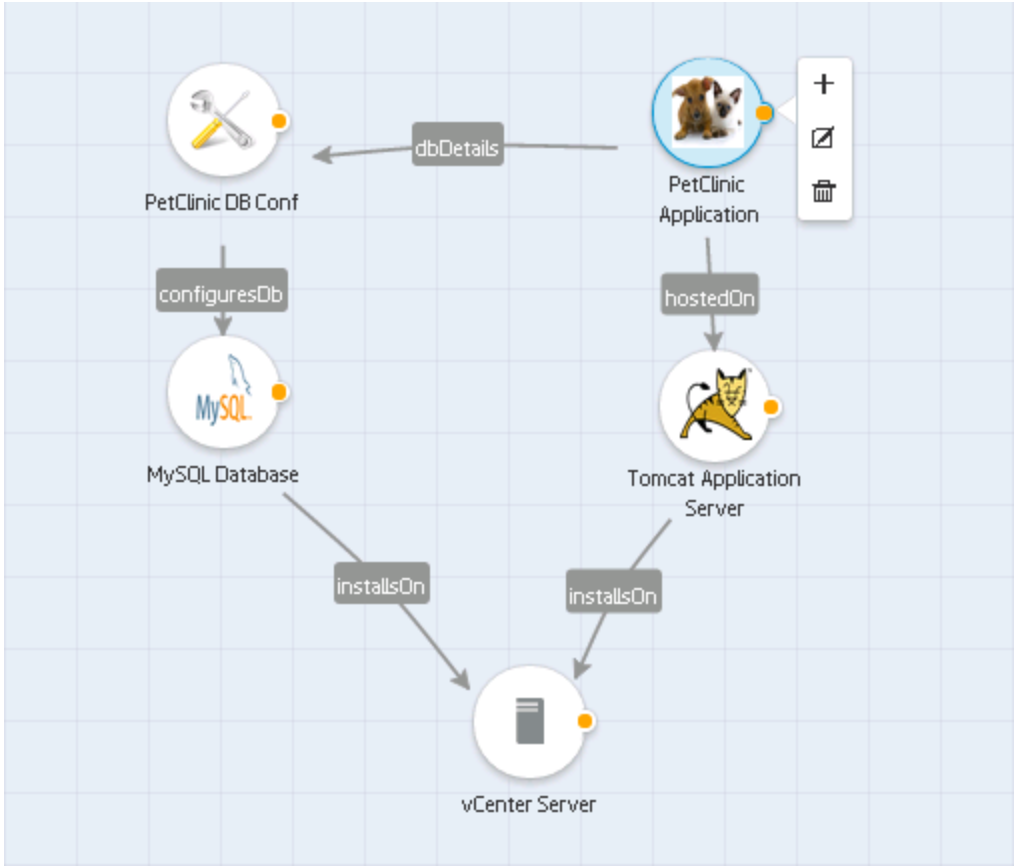
Hewlett Packard Enterprise Codarは、システムへのあらゆる変更がリリース可能で、すべてのコード変更を運用環境にデプロイ可能な継続的デリバリを促進します。Codarは、継続的デリバリの自動化を実現します。その結果、コード変更があると、ビルドが開始されデプロイされます。自動化されたユニットテストも実行されます。アプリケーションは、ランブック自動化フローで定義されたポリシーに基づいて、環境に自動的にデプロイされます。継続的デリバリの目的は、デリバリを頻繁に行い、ユーザーからのフィードバックを迅速に得ることです。



宣言ベースのモデリング

宣言ベースのモデリングを使用してアプリケーション開発を自動化する場合、ユーザーはアプリケーション開発の終了状態 (アプリケーションのコンポーネントと、コンポーネント間の依存性) を宣言できます (その状態に至るプロセスは、バックグラウンドで開始されます)。こうして、ユーザーは、どのようにデプロイするかではなく、何をデプロイするかに集中できます。その結果、マルチティアアプリケーションのデプロイメントを短時間で自動化できます。長期にわたる管理も大幅に簡素化されます。

Codarは、ユーザーインタフェースを使って複雑なデザインの作成、統合、管理を行う宣言ベースのモデル開発をサポートしています。モデルは、トポロジデザインとそのプロパティで構成されます。Codarでは、プロパティを具現化しているときに柔軟に変更できます (レイトバインディングと同様です)。



トポロジ構成

アプリケーションデザイン (トポロジデザインともいう) は、コンポーネントとコンポーネント間の関係を指定することで、アプリケーションのライフサイクルを定義します。アプリケーションデザインは、ライフサイクルシーケンスをクラウドプロバイダーに委任します。

アプリケーションデザインには、次の2つのタイプがあります。

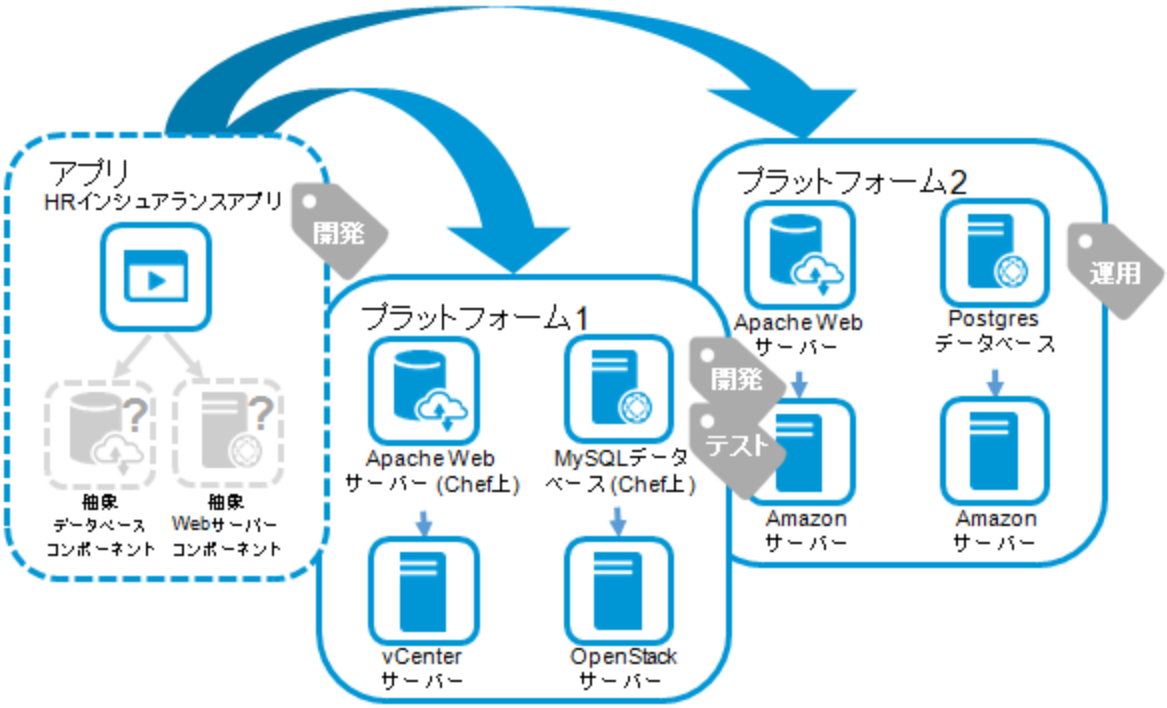
- 完全デザイン: このデザインにフルフィルメントに必要なすべてのコンポーネントが存在する
- 部分デザイン: このデザインの他にフルフィルメントに必要なデザインが存在する

トポロジ構成は、実行時に、インフラストラクチャーデザインを使ってアプリケーションデザインを構成する際に使用します。アプリケーションデプロイメントでは、デプロイメントごとにインフラストラクチャーのニーズは異なります。このように、アプリケーションデザインで必要とされるさまざまなインフラストラクチャーを定義する作業ではトポロジ構成が役立ち、デプロイメント時にさまざまなインフラストラクチャーデザインを使った構成が可能になります。

コンポーネントの記述には、機能と特性が使用されます。アプリケーションデザインは、デザインで機能コンポーネントと特性を使用して、要件を定義します。アプリケーションデザインはそれだけではプロビジョニングできず、互換性のあるサービスデザインを選択する必要があります。サービスデザインコンポーネントでは、機能および特性に基づいて互換性チェックが実行され、適合するデザインが互換サービスデザインとしてデプロイメント時に選択されます。

次の図は、HRインシュアランスアプリケーションのトポロジ構成を示しています。このアプリケーションはデータベースコンポーネントとWebサーバーコンポーネントを必要とします。これは、アプリケーションデザインアプリで定義されています。実行にはプラットフォーム1が使用されます。ここでは、Webサーバーの機能および特性を持つApache

Webサーバーと、データベースの機能および特性を持つMySQLデータベースが稼働しています。同様に、プラットフォーム2もアプリ要件を満たしています。



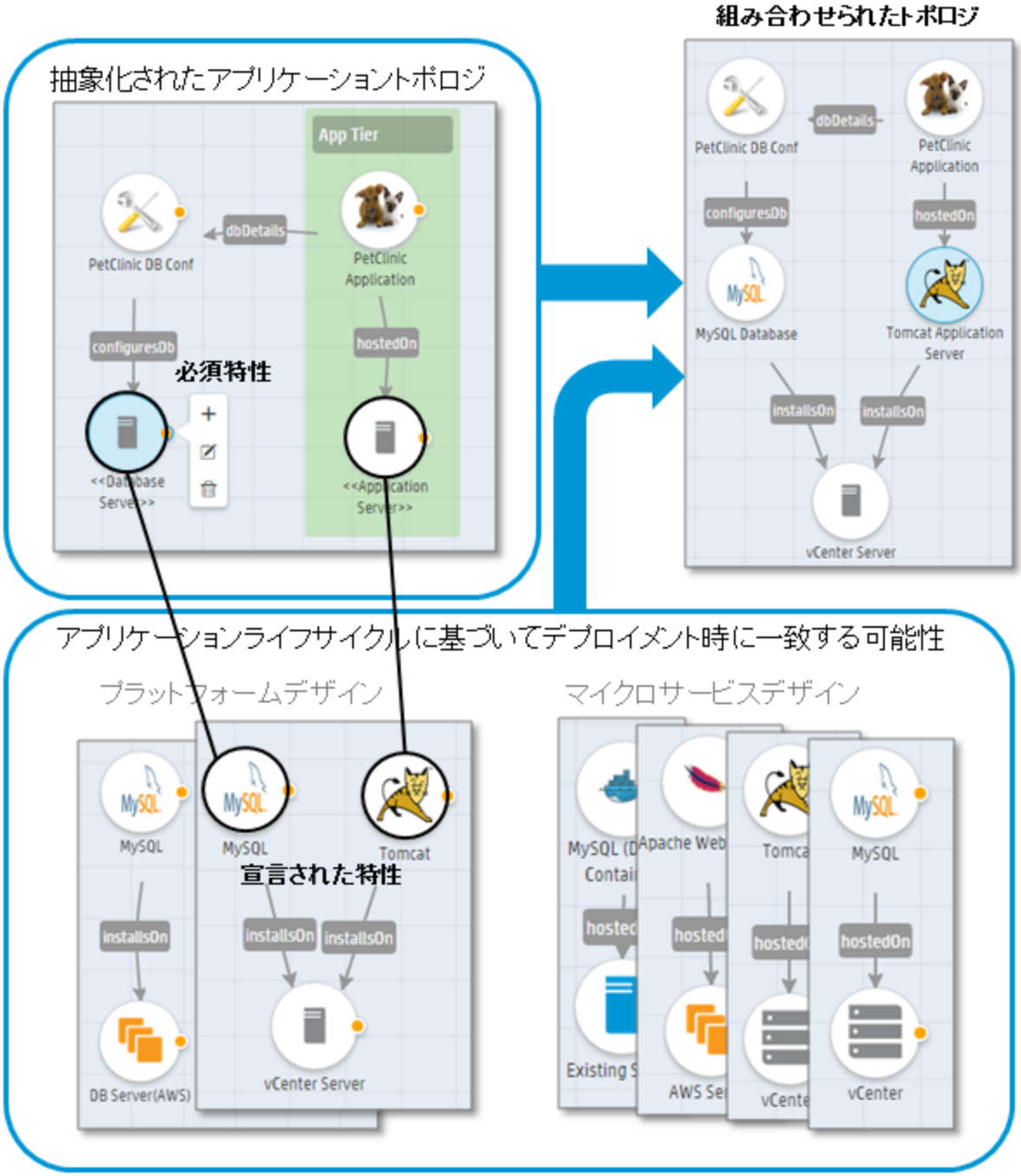
マイクロサービス

単一のサービスではなくプラットフォームまたはインフラストラクチャーサービスを提供する複数のインフラストラクチャーデザインを使用して部分的なアプリケーションをデプロイできます。複数のオープン要件 (機能) がある部分デザインを複数のインフラストラクチャーデザインを使用して構成することで、すべてのオープン要件を満たして、アプリケーションデザイン全体を構築することができます。すべての機能と一致する単一のデザインを選択することも、異なる複数のデザインからのコンポーネントを選択することもできます。

複数のオープン要件 (機能) がある部分デザインを複数の (マイクロ) サービスデザインを使用して構成することで、すべてのオープン要件を満たして、アプリケーションデザイン全体を構築することができます。

たとえば、アプリケーションにデータベースサービスとアプリケーションサービスが必要な場合、単一デザイン内にデータベースとアプリケーションがあるデザインを選択することも、1つのサービスデザインからデータベースを選択し、別のサービスデザインからアプリケーションを選択することもできます。

マイクロサービスの選択に基づいて実行時に組み合わせられたトポロジが作成されます。マイクロサービスはライフサイクルのステージに関連付けることができます。



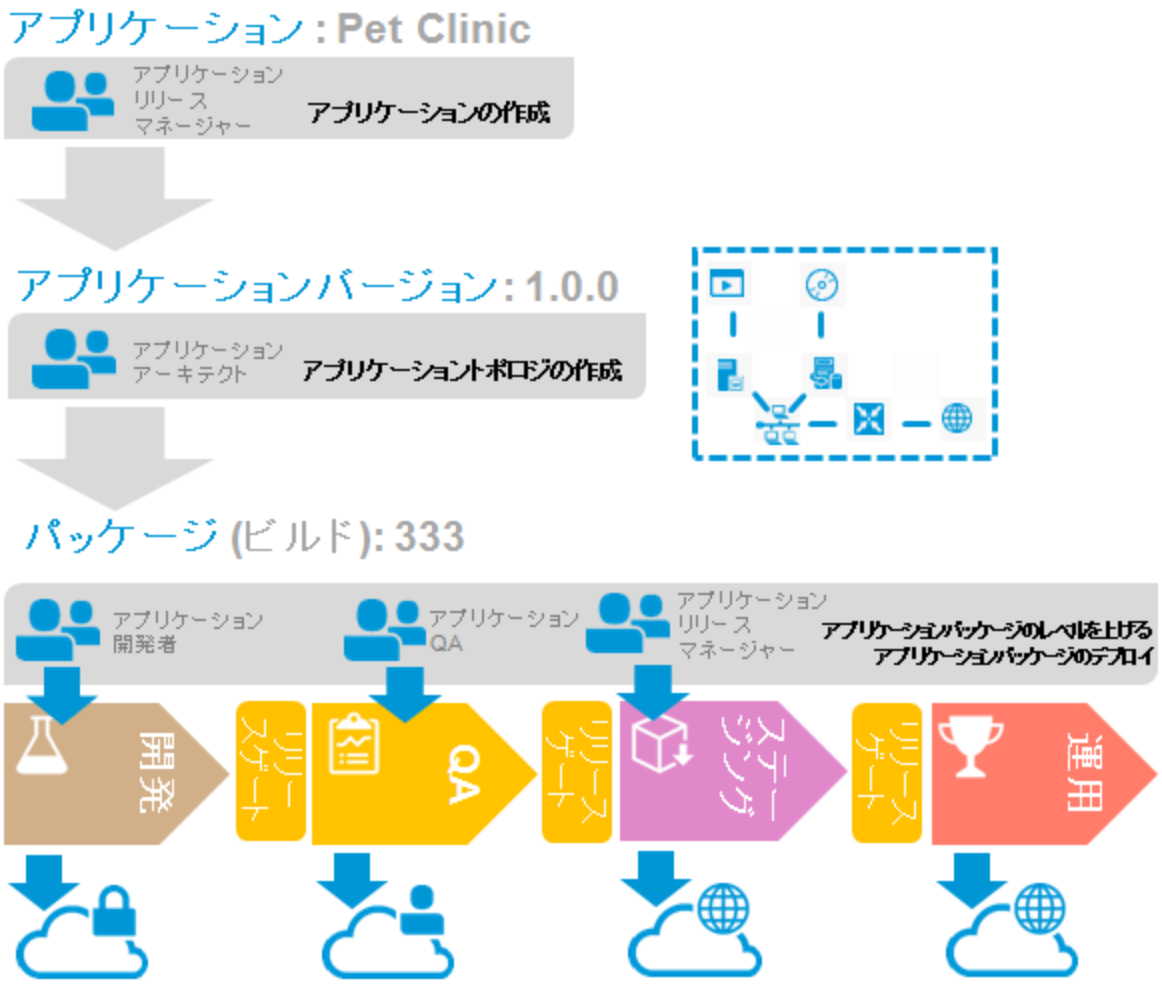
アプリケーションパイプライン管理

アプリケーションのデプロイメントの自動化は、複雑で時間を要するプロセスであり、多くの調査を必要とします。開発と運用でアプリケーションのデプロイ方法が異なると、多くのエラーの原因になります。アプリケーションパイプライン管理を使用すると、同じトポロジモデルを使用して異なる複数の環境にアプリケーションをデプロイできます。各ステージで異なるマイクロサービスを選択できますが、アプリケーションデザインは変わりません。つまり、異なるライフサイクルステージを通じて同じデザインがデプロイおよびテストされます。

また、リリースパイプラインをカスタマイズして、アプリケーションチームごとに別々のライフサイクルステージを使用することもできます。これにより、完全に自動化された連続デプロイメントが可能になります。Codarは、手動のステップを解消することにより、アプリケーションデプロイメントの品質向上とコスト削減を実現し、同時にアプリケーションリリースサイクルのアジリティを向上させます。

Codarでのパイプライン管理には、次の内容が含まれます。

- 独自のロールを作成することで、独自のユーザーアクセス構造を構築できるようにする
- 事前定義のライフサイクルステージに加えて、独自のライフサイクルステージを作成する
- すでに存在しているリソース環境を選択して、特定のライフサイクルステージのみに関連付けることで、事前定義のライフサイクルステージのサブセットを含むライフサイクルステージのスーパーセットを作成する
- パイプライン統計を表示し、デプロイメントを視覚的に表現する
- パッケージ、アクション、環境に基づいて表示をフィルター処理する



パッケージの管理

パッケージはアプリケーションデザインのスナップショットを表し、デザイン内でのプロパティのパラメーター化を可能にします。パッケージはアプリケーションの特定のビルドを表すと言うこともできます。

パッケージはアプリケーションに関するデプロイ可能な最小単位です。これは、実装アーティファクト (アプリケーションがデプロイされる方法) と、デプロイメントアーティファクト (war、earなど、デプロイされるライブラリの場所) の両方を表します。

パッケージは、ライフサイクルステージに関連付けられます。1つのパッケージは、開発 (Development)、テスト (Testing)、ステージング (Staging)、運用 (Production) のいずれかのステージに属することができます。

パッケージはパイプライン管理に関連付けられます。パッケージは、特定のステージでの昇格や拒否など、ライフサイクルステージをまたいで管理できます。たとえば、QAロールを持つユーザーは、パッケージを拒否できます。

タスク

- **特定のアプリケーションバージョンからパッケージを作成** -アプリケーションバージョンは、複数のパッケージから構成される場合があります。
- **パッケージのデプロイまたは再デプロイ** -この場合、アプリケーションデザインの対応する状態と、パッケージに指定されたデザインのプロパティのフルフィルメントが行われます。
- **パッケージの削除** -[リリースパイプライン] タブを開き、Ctrlキーを押しながらパッケージを複数選択し、[**削除**] をクリックします。

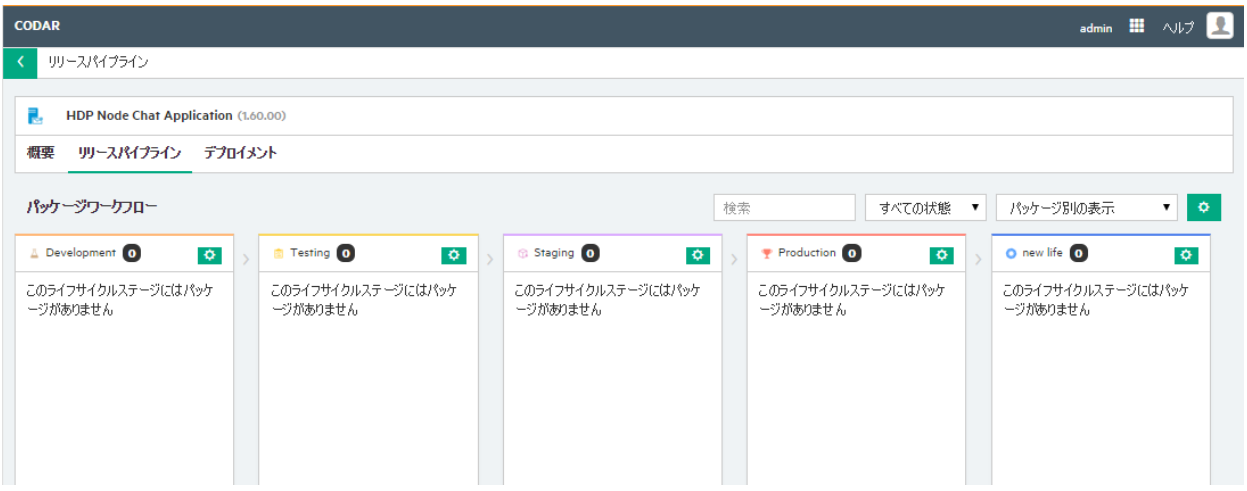
  インスタンスが関連付けられているパッケージは削除できません。

パッケージの操作

Codarは、DevOps環境を実装するための一元化された構造です。さまざまなロールで、パッケージのデプロイ、再デプロイ、レベルを上げる、拒否を行うことができます。パッケージを、一貫した再現性のある方法で、1つのステージから次のステージへレベルを上げます。このようにして、アプリケーションが運用状態に進むときの可視性がチームメンバーに保証されます。

パッケージを作成してデプロイすると、新しい仮想マシンが作成され、パッケージがデプロイされます。デプロイ済みのインスタンスについてテストを実行し、パッケージのレベルを上げるか拒否することができます。

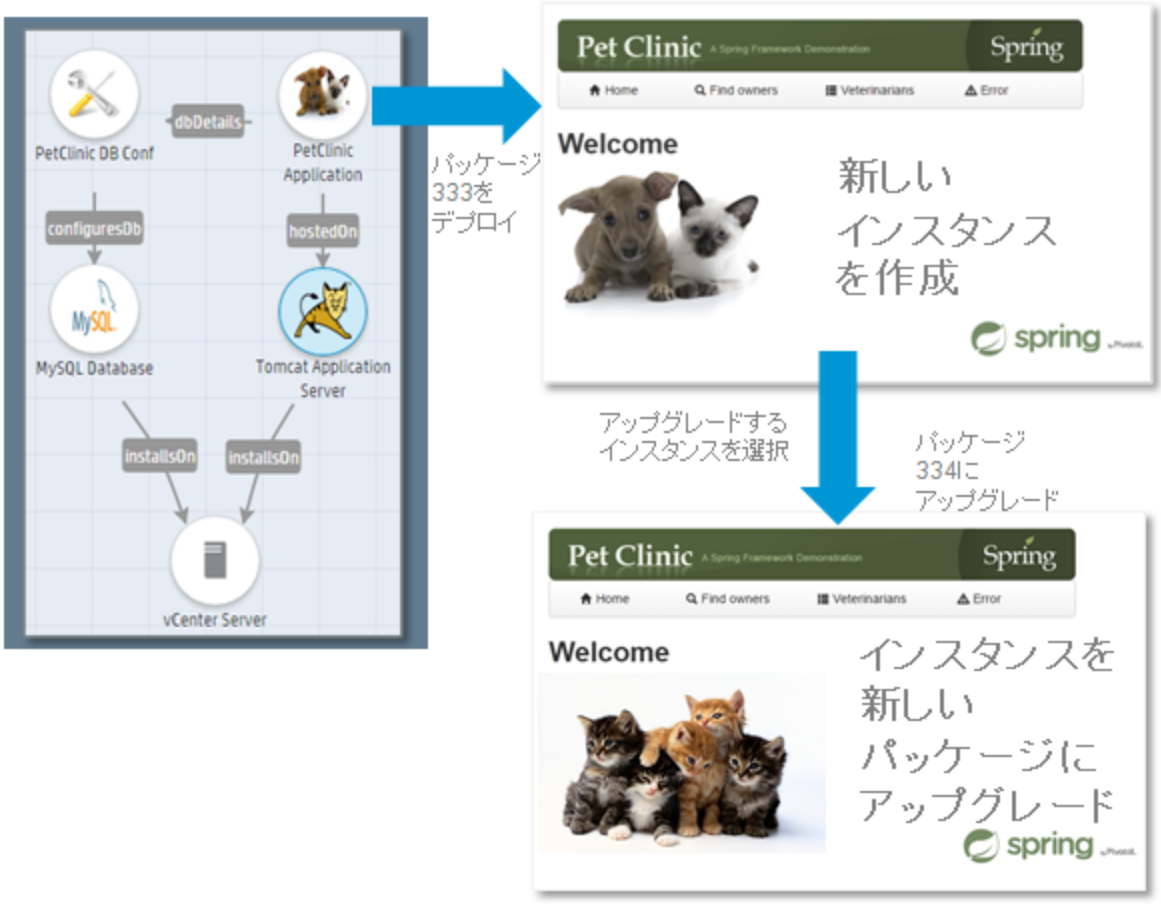
次のスクリーンショットに示すように、Codarはアプリケーションパイプライン管理機能を促進します。



デプロイと再デプロイ

古いパッケージがデプロイされているインスタンス上にパッケージを再デプロイすることができます。インスタンスの詳細を表示し、既存の変わらない要求を選択できます。再デプロイを使用して、コンポーネントをアップグレードし

たりパッチを適用したりすることもできます。再デプロイにより、すべてのコンポーネントの変更アクションが開始されるので、デザイン内のすべてのコンポーネントを新しいバージョンにアップグレードすることができます。

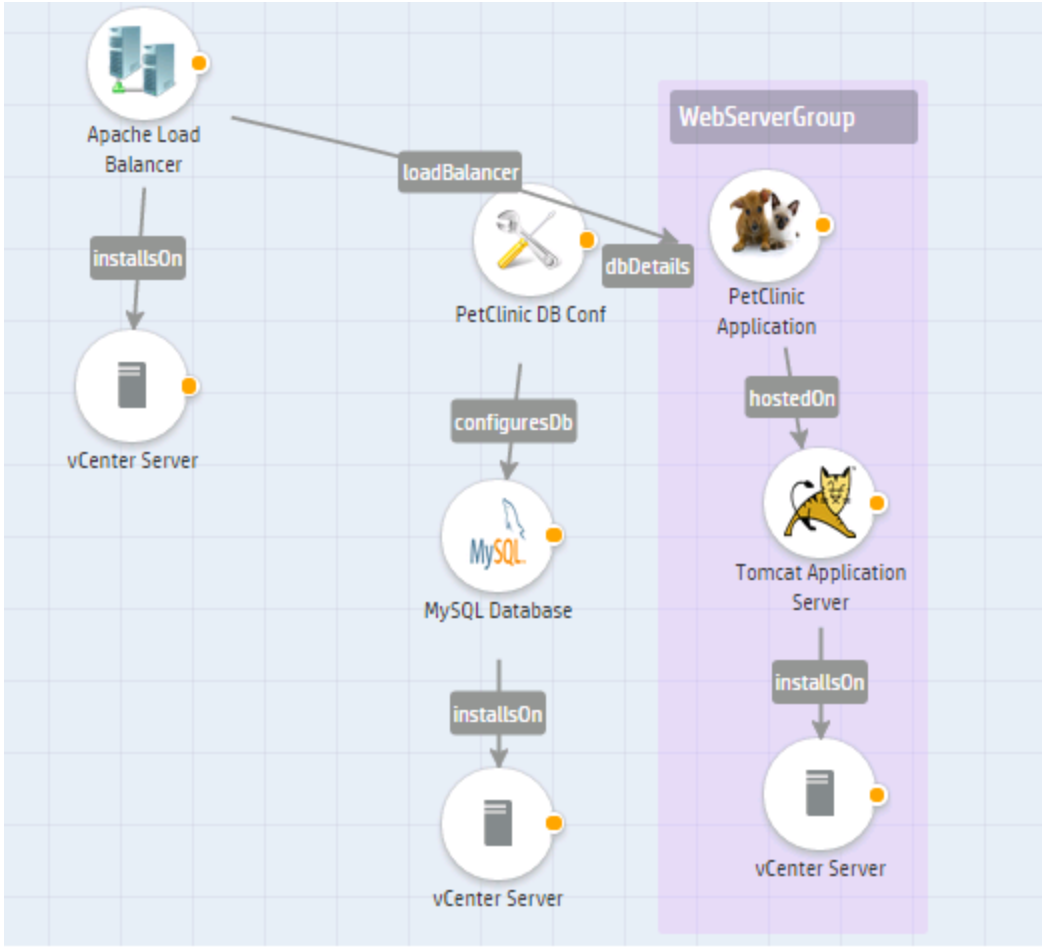


スケールアウト

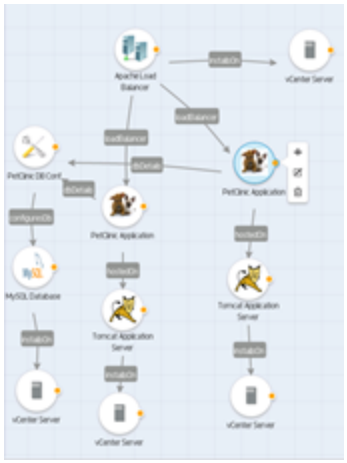
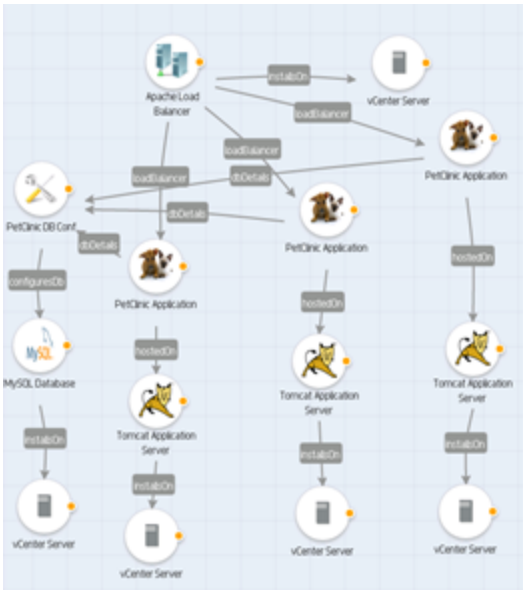
トポロジデザインの作成中に、スケーリンググループを作成できます。スケーリンググループは、スケーラブルスタックを表します。1つのアプリケーションデザイン内で複数のスケーラブルグループを使用できます。

デプロイメントが完了した後でスケールアウトすることができます。スケールアウトすると、スタック全体が複製されます。

たとえば、下の図は、webServerGroupという名前の論理的なグループとしてのWeb層を示しています。



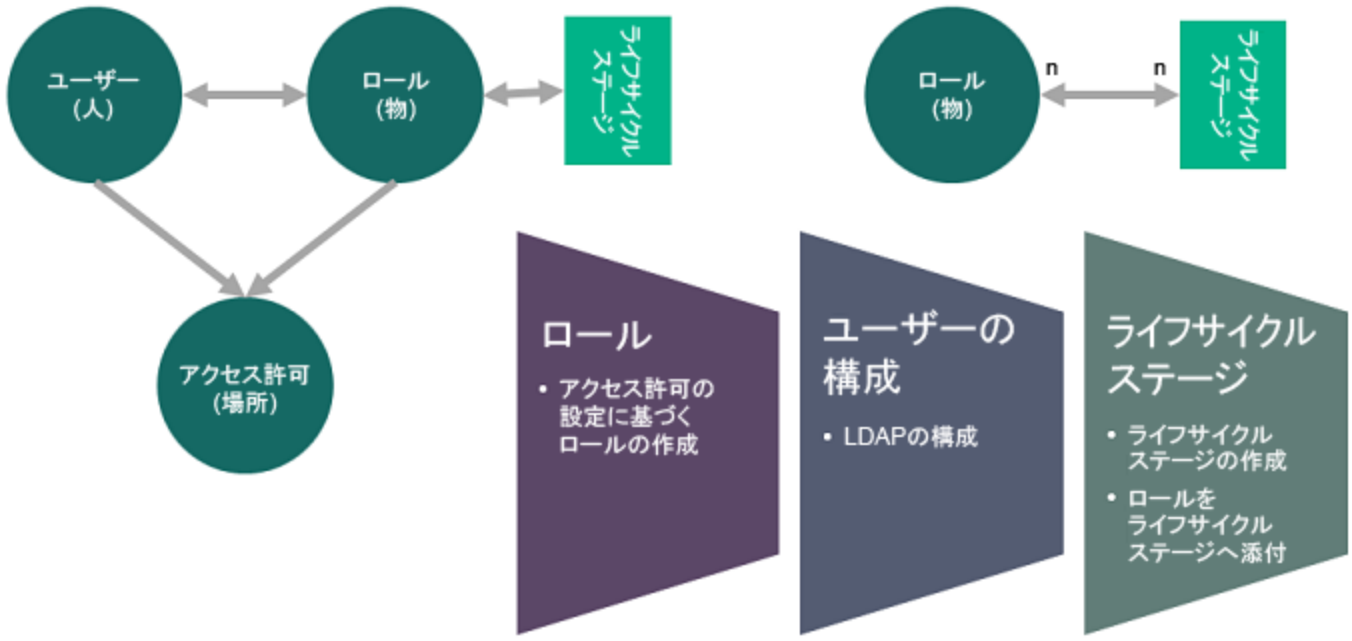
このグループは、開 発 ステージで1つのグループにスケールインされ、テスト ステージで下 の図 のように2つのグループにスケールされます。

開 発 ステージ	テスト ステージ
	

ロールとユーザーアクセス

ユーザーアクセスは、トポロジデザインで構成できます。ユーザーとLDAPグループの両方をデザインに追加できます。アプリケーションアーキテクトは、デザインを作成し、デザインをパブリックにするか、特定のユーザーに制限することができます。

Codarでは、ユーザーアクセスはロールとアクセス許可で構成されます。Codarのすべてのユーザーには、1つまたは複数のロールが割り当てられます。すべてのロールには、1つまたは複数のアクセス許可が割り当てられます。そのため、特定のロールに属するユーザーは、そのロールに対して定義されたすべてのアクセス許可を持っています。Codarにはいくつかのロールが事前定義されていますが、ユーザーは独自のロールを作成して、作成したロールにアクセス許可を割り当てることができます。カスタムロールの作成方法については、Codarオンラインヘルプを参照してください。



管理者とアプリケーションアーキテクトは、次のようにユーザーとグループを構成できます。

- 各ロールのユーザーは、細かく制御できるようにアプリケーションレベルで定義されます。
- グループは、アプリケーションに対するロールを自動的に割り当てるためのアプリケーションチームを表す必要があります。

次の図は、アプリケーションアーキテクト、開発者、QA、リリースマネージャーを含むさまざまなユーザー用に構成されたデザインを示しています。


CODAR
admin
ヘルプ
トポロジデザイン
New デザイン
アプリケーション
概要
バージョン
アクセス制御
アクセス制御は、このデザイン上
デザインの作成または更新が可能なす
管理
理」ボタンを使用します。
アクセス制御の管理
このデザインへのアクセスを制限しないでください
次のユーザーまたはグループに対して、このデザインへのアクセス制御を制限します
検索
下の [追加] ボタンを使用して、ユーザーとグループがこのデザインにアクセスできるようにしてください
+
保存
キャンセル

ライフサイクルのステージおよびアクション

ライフサイクルアクションでは、HP Operations Orchestrationなどのプロセスエンジンを利用してサービスプロバイダーと通信することにより、サービスの初期デプロイメントが行われます。これ以外にも、要求に応じたサービスの変更やデプロイメントからのサービスの削除など、重要な機能を実行するライフサイクルアクションがあります。

すべてのライフサイクルはステージで構成され、すべてのステージにはロールが関連付けられています。つまり、ある特定のライフサイクルステージ中は、そのステージのロールに所属するすべてのユーザーが、そのロールに定義されているオペレーションを実行できます。たとえば、開発ライフサイクルステージにApplication Architectロールが関連付けられている場合、Application Architectロールに所属するユーザーは、開発ライフサイクルのロールに関連付けられているタスクを実行できます。

次に、Codarの事前定義済みのライフサイクルステージを示します。

- 開発 (Development): これは通常、最初のステージであり、コードが開発され、アプリケーションアーティファクトが作成されます。
- テスト (Testing): 開発ステージで開発されたコードに対して、テストケースが実行されるステージです。
- ステージング (Staging): 運用環境を複製する運用前ステージであり、コードとアーティファクトのテストに使用します。
- 運用 (Production): これは通常、最終ステージであり、アプリケーションが稼働環境にデプロイされます。

事前定義済みのライフサイクルステージの他に、ユーザーが作成できるカスタムステージもあります。カスタムステージの作成、編集、削除については、Codarのオンラインヘルプを参照してください。

次の表に、各ライフサイクルステージで行われるパッケージに関係するアクションを示します。

ステージ	レベルを上げる	デプロイ、再デプロイ	編集	削除	拒否
最初のステージ (通常は開発ステージ)	○	○	○	○	○
中間	○	○	○	○	○
最終ステージ (通常は運用ステージ)	×	○	○	○	○

ライフサイクルステージは、[**リリースオートメーション**] > [**パイプライン構成**] タイルを使用してアクセスできます。

次のアクションを使用して、パッケージをあらゆるステージでデプロイまたは移動します。これらのアクションは、リリースゲートアクションが定義されていないときのフローを示しています。

- **レベルを上げる**: パッケージを次のライフサイクルステージに移動します。パッケージの状態は、アクティブなままに保たれます。
- **デプロイ、再デプロイ**: パッケージをデプロイします。
- **編集**: パッケージのプロパティを変更します。
- **拒否**: パッケージを他のステージに進めないようにします。パッケージは、現在のステージに留まり、その状態は拒否済みに設定されます。アクションボタンは、使用できなくなります。
- **削除**: パッケージの削除 -パッケージはシステムから恒久的に削除されます。パッケージを削除できるのは、デプロイ済みの関連するすべてのインスタンスがキャンセルまたは削除されている場合のみです。
- **更新**: パッケージの現在のステータスを取得します。

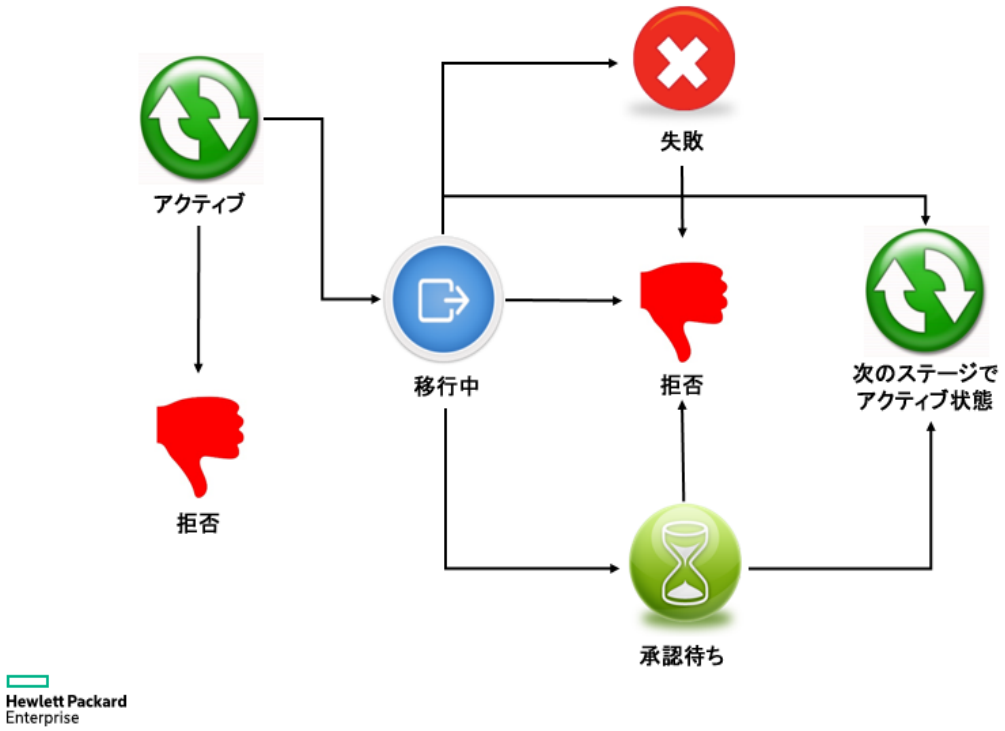
パッケージの状態

パッケージには、次の状態があります。

- アクティブ: 現在のライフサイクルステージでパッケージはアクティブ状態
- 拒否済み: パッケージは拒否され、次のライフサイクルステージには移動しない
- 移行: パッケージは次のライフサイクルステージに移行中
- 失敗: パッケージの昇格に失敗した

パッケージを拒否すると、そのパッケージは現在のステージに保たれ、その状態は拒否済みに設定され、それ以降はアクションを適用できません。ただし、パッケージを削除することは可能です。パッケージはシステムから削除されます。

パッケージのレベルを上げると、パッケージは次のステージに移動し、アクティブ状態に留まります。パッケージは常に最初のライフサイクルステージで作成されます。Codar Jenkinsプラグインが構成されている場合、ビルドが成功すると、JenkinsプラグインがCodarと通信してパッケージを作成します。



ライフサイクルステージによるインフラストラクチャーデザインのグループ化

アクティブパッケージの部分デザインでは、パッケージのデプロイウィザードでプロビジョニングを行う際に、インフラストラクチャーデザインを選択する必要があります。インフラストラクチャーデザインはライフサイクルステージごとにグループ化することが可能です。これにより、ライフサイクルステージでパッケージをデプロイする際、そのライフステージで指定されたインフラストラクチャーデザイングループのみを表示できます。

特定のライフステージ用にインフラストラクチャーデザインをグループ化するには、各トポロジデザインで、ライフサイクルステージの名前でタグを作成します。ライフサイクルステージの作成時には、ライフサイクルステージのタグが

自動的に作成されます。そのため、デザインを適切なライフサイクルステージタグに関連付けるだけで済みます。たとえば、開発タグを作成し、開発ライフサイクルの状態で必要なデザインに関連付けることが可能です。

注: [テスト] タブでテスト実行ウィザードを実行する場合、タグでグループ化されたデザインではなく、すべてのデザインが表示されます。

リリースゲートアクション

リリースゲートアクションはユーザー定義のアクションで、ライフサイクルステージ間の昇格要求チェックの役割を果たします。パッケージが有効化された各アクションを通過し、ライフサイクルステージのすべてのアクションのステータスが成功の場合にのみ、パッケージが次のステージに昇格されます。

リリースゲートアクションには次の3つのタイプがあります。

- デプロイアクション

  このアクションは、事前定義済みのOperations Orchestration (OO) コンテンツパックに基づいて、アプリケーションデザインの一部または全体をデプロイします。デプロイアクションは、昇格要求を開始したユーザーに昇格の成功または失敗を通知する電子メールメッセージが送信されるように構成できます。デプロイアクションの実行に失敗し、パッケージが昇格されなかった場合、ユーザーはパッケージを拒否してデプロイメントをクリーンアップすることを選択することもできます。

  デプロイアクションの作成、編集、削除については、Codarのオンラインヘルプを参照してください。

- カスタムアクション

  このアクションでは、デプロイアクションを使用して作成されるパッケージのデプロイメントが必要です。カスタムアクションは、デプロイ済みインスタンス上でOOフローを実行します。デプロイアクションはカスタムアクションへの入力として使用されます。そのため、カスタムアクションを作成する前に、デプロイアクションが存在しているか、デプロイアクションを作成しておく必要があります。

  カスタムアクションは、昇格要求を開始したユーザーに昇格の成功または失敗を通知する電子メールメッセージが送信されるように構成できます。カスタムアクションの実行に失敗した場合、ユーザーはパッケージを拒否するか、パッケージの昇格を続行するかを選択できます。

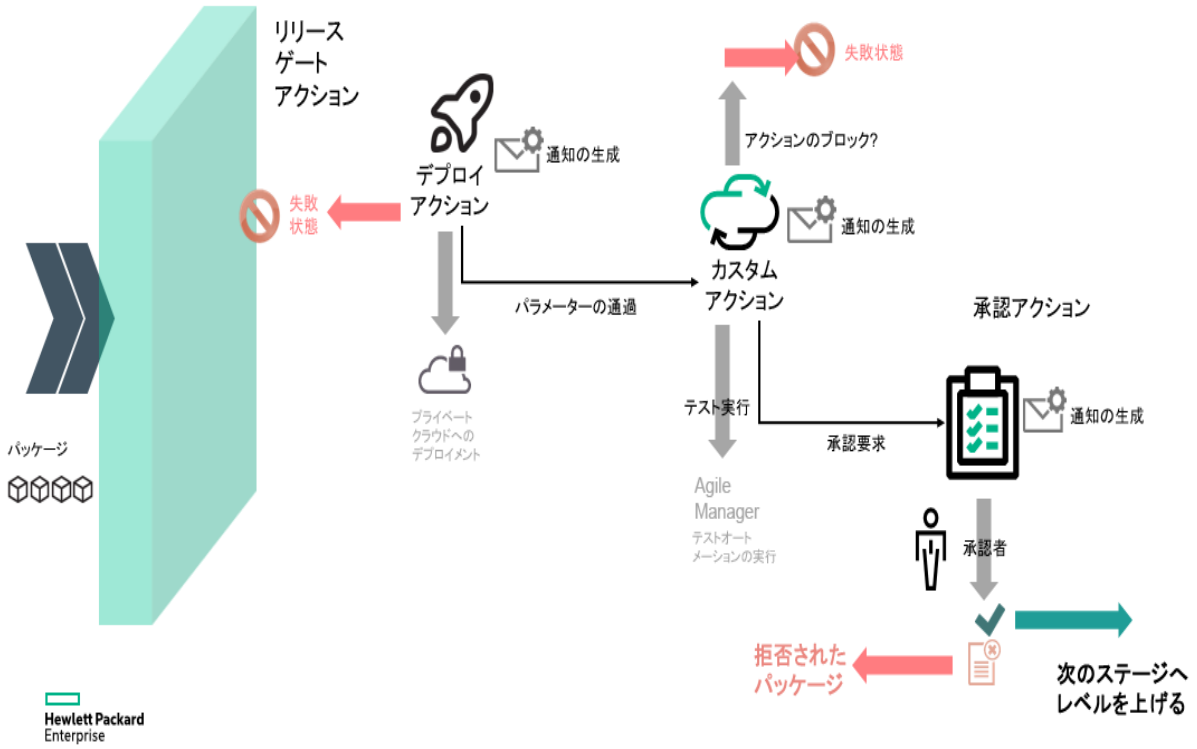
  カスタムアクションの作成、編集、削除については、Codarのオンラインヘルプを参照してください。

- 承認アクション

  このアクションは、指定された承認者がパッケージの昇格を手動で承認または否認する場合にのみ、パッケージを昇格します。承認アクションは、パッケージの昇格を自動的に承認または拒否するように構成することもできます。

  承認アクションの作成、編集、削除については、Codarのオンラインヘルプを参照してください。

次の図は、リリースゲートアクションの仕組みを表したものです。

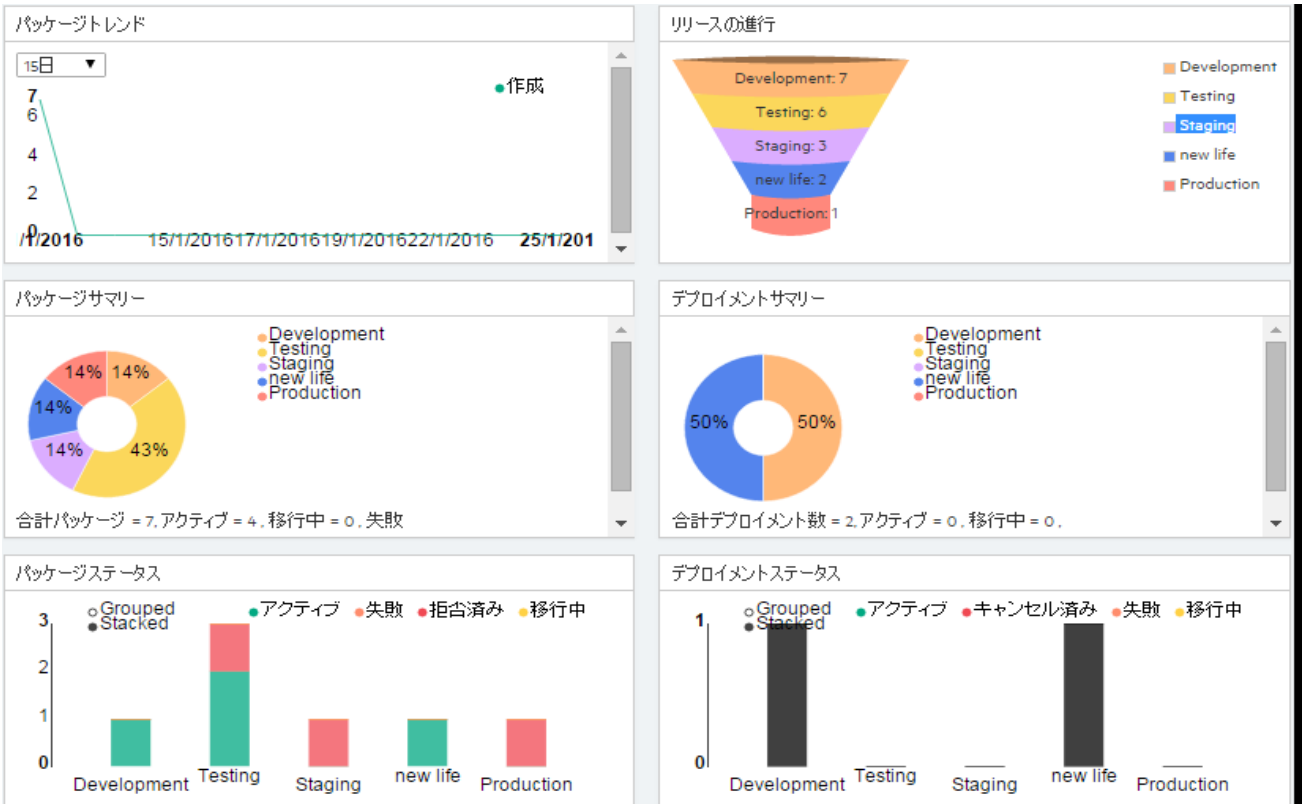


パイプライン統計

[パイプライン統計]タブには、パッケージの詳細情報と、パッケージサマリー、トレンド、状態、デプロイメントステータスなどのグラフィカル表現が表示されます。ここでは、すべてのパッケージとデプロイメントの全体像が提示され、パッケージのデプロイメントについて情報に基づいた意思決定を実行できます。

任意の日に作成されたパッケージの数、成功した移行の数、デプロイメントの数などの情報が表示されます。

パイプライン統計の詳細については、Codarのオンラインヘルプを参照してください。



環境

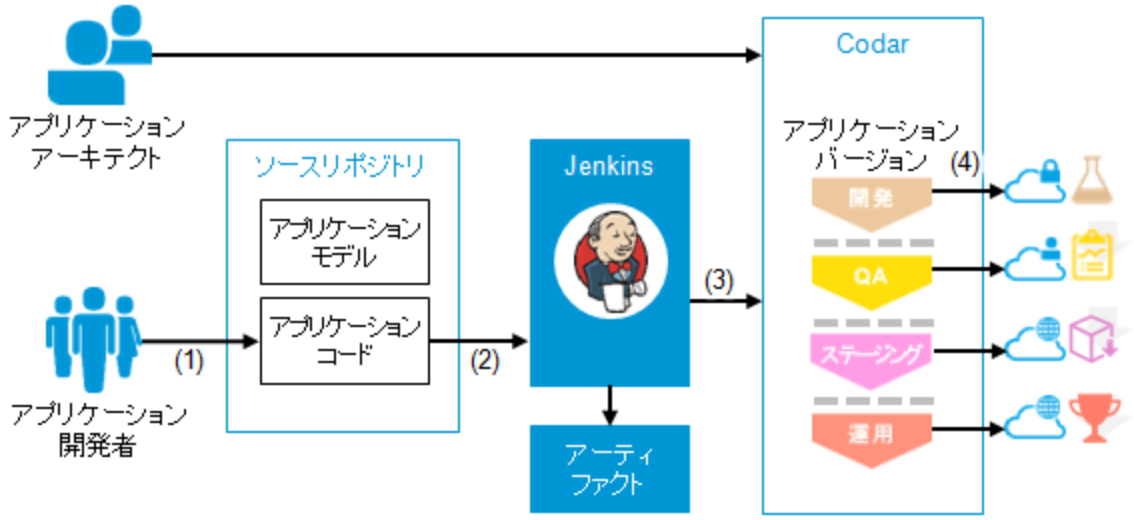
各ライフサイクルステージに合わせてアプリケーションレベルで異なる環境を選択できます。たとえば、デプロイメント用にvCenterを構成し、ステージング用にはパブリッククラウド環境を選択することができます。

外部統合

Codarは、オープンで拡張可能です。これは、Jenkins、Hudsonなどのさまざまなビルドシステムと統合できます。REST APIの包括的なセットを他の外部ツールと併用すると、継続的な統合、デプロイメント、デリバリを実現できます。Codarのアーキテクチャーには、DevTestおよびDevOpsのカスタマイズ済みフローにフックするオプションも用意されています。

Jenkinsの統合

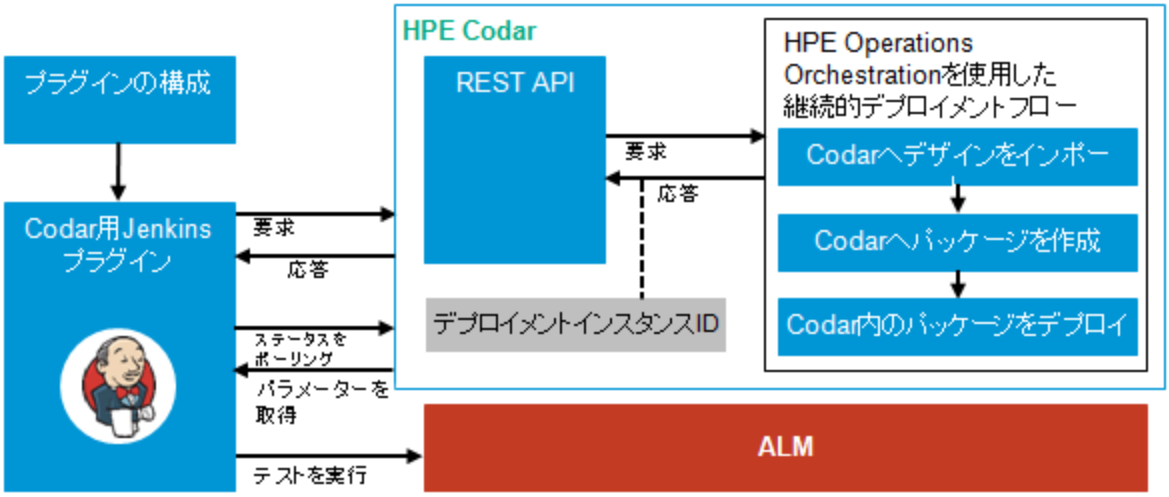
Codarには、継続的デプロイメントのためのJenkinsプラグインが含まれています。次の図は、その仕組みを示しています。



1. 開発者が変更をチェックインします。
2. 継続的統合によってビルドが開始されます。
3. Jenkinsプラグインがパッケージを作成してデプロイします。
4. ライフサイクルステージでアプリケーションが異なる環境にデプロイされます。
5. 連続昇格の場合、すべてのリリースゲートアクションが正常に実行されると、パッケージは最終ライフサイクルステージに移動します。

ALMの統合

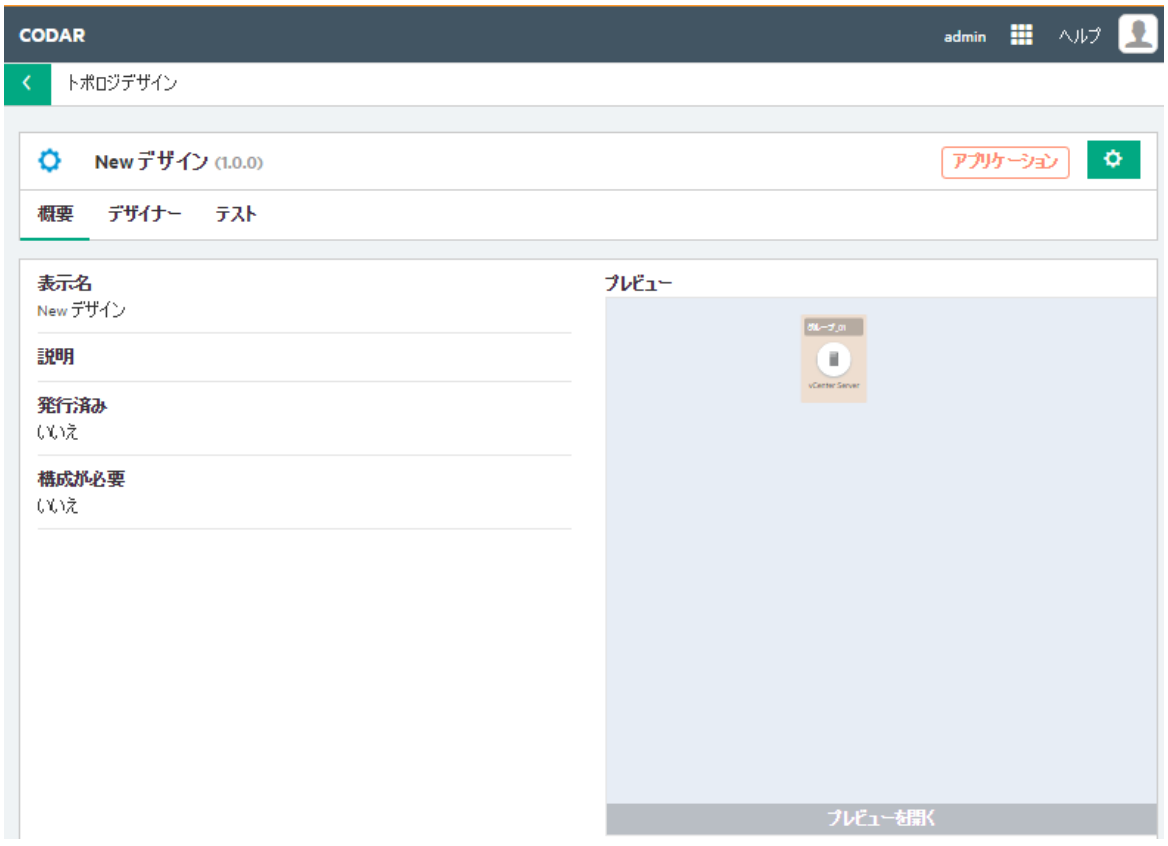
デプロイメントが成功した後に、Application Lifecycle Management (ALM) とCodarを統合してテストを実行できます。次の図は、Jenkinsが調整機能としてどのような処理を行うかを示しています。



Infrastructure as code (IaaC)

IaaC (Infrastructure as Code) を管理すると、ITチームは、インフラストラクチャーとアプリケーションのプロビジョニング方法のコードレビューやユニットテストのコードを開発する際にベストプラクティスを利用できます。

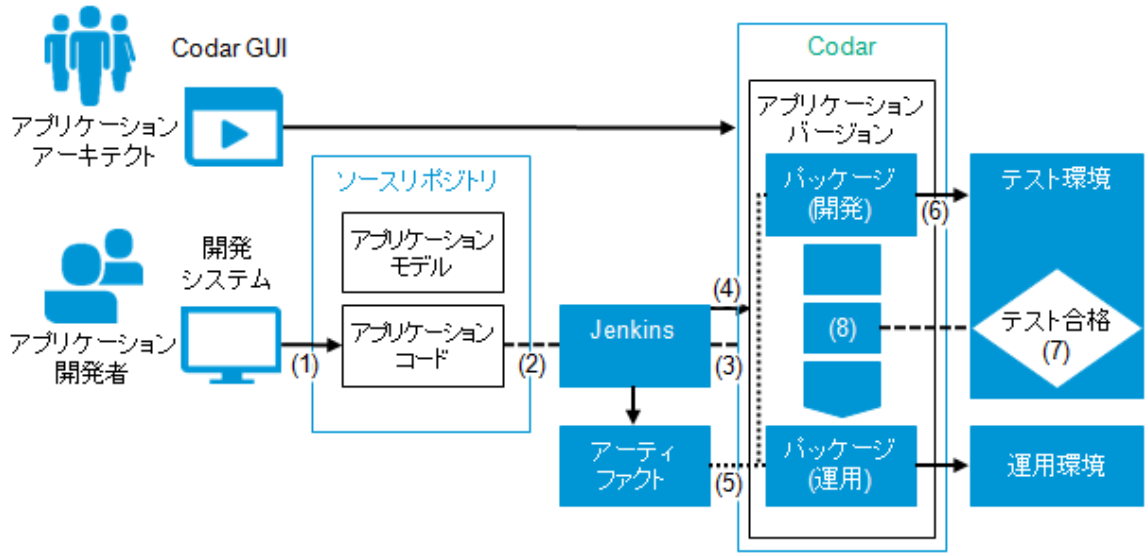
Codarは、インフラストラクチャーをコードとして管理できます。サーバー構成、ネットワーク、ボリューム、関係、アプリケーション固有の詳細 (アプリケーションバージョン、パッケージ情報など) を含むことができるトポロジデザインは、JSON形式でエクスポートし、ソース管理システムのアプリケーションで管理できます。開発者は、テキストエディタを使用してモデルに変更を加え、自動化のために使用できます。変更したモデルをインポートして、Codarに戻すこともできます。



ユースケース: 継続的な統合、デプロイメント、デリバリ

目的は、アプリケーションを継続的統合 (CI) と継続的デプロイメントに対応させることです。アプリケーション開発者はアプリケーションをコード化し、アプリケーションアーキテクトはアプリケーションをCodarインタフェースでモデル化し、そのアプリケーションモデルをコード (IaaC) としてエクスポートします。アプリケーション開発者がコードをチェックインすると、Jenkinsビルドがトリガーされ、アプリケーションモデルを使用してアプリケーションが特定の環境にデプロイされます。アプリケーションのデプロイが終了すると、継続的開発プロセスが継続的デリバリに拡張されますが、デプロイされたインスタンスでアプリケーション固有のテストを自動的に実行できます。このとき、テストの結果によっては、アプリケーションが別の環境にデプロイされる可能性があります。

次の項では、このシナリオがCodarでどのように実現されるかを説明します。



アプリケーションのモデル化

アプリケーションアーキテクトは、アプリケーションをグラフィカルにモデル化します。これは、デザイナーインタフェースで、デザインの必要コンポーネントを追加し、それを関係を使って接続して行われます。Codarには、標準的なコンポーネントのパレットがあります。また、HPE Operations Orchestration、Chefなどの各種開発エンジンからコンポーネントをインポートする (取り込む) こともできます。このようなデザインはアプリケーションモデルと呼ばれ、アプリケーションのデプロイ方法を表現します。アプリケーションモデルは、JSON形式でエクスポートして、外部のソースリポジトリで管理できるため、IaaC (Infrastructure as Code) を実現できます。

継続的な統合とデプロイメント

継続的統合では、サンプルアプリケーションのコードと、アプリケーションをデプロイするためのモデル (JSON形式) は、ソースリポジトリにあります。

アプリケーションの開発者がアプリケーションのコード変更を行い、そのコードをソースリポジトリにチェックインすると (1)、Jenkinsによりビルドが開始されます (2)。

Codarによって、CodarのIPアドレス、ユーザー名、パスワードなどの詳細情報を持つJenkinsプラグインが提供されます。接続がポストビルドステップの一環として確立され、APIが起動されます (3)。次に、そのAPIによって、各種アクションを実行するワークフローが起動され、継続的開発と継続的デリバリが実現されます。

アプリケーションデザインのインポート

アプリケーションモデルがCodarにまだインポートされていない場合や、アプリケーションモデルに変更があった場合、継続的デプロイメントワークフローは、それをアプリケーションデザインの新バージョンとして、JSON形式 (IaaC) でCodar (4) にインポートします。これにより、アプリケーションの開発者とアーキテクトが行った変更を、デプロイメント時に考慮することができます。

アプリケーションモデルがすでにインポート済みか、またはアプリケーションデザインに変更がない場合、このインポートオペレーションは実行されず、Codar内のアプリケーションバージョンは同じままに保たれます。この点は、注意してください。アプリケーションモデルは、デザイナーの [トポロジ] タイルに表示できます。

環境へのデプロイ

パッケージが作成されると、継続的デプロイメントワークフローが、環境に基づいてアプリケーションデザインのフルフィルメントを実行します (6)。パッケージのデプロイメントは、[デプロイメント] タブに表示できます。

ランブック自動化エンジンは、インフラストラクチャーレイヤー、プラットフォームフェイルオーバイヤー、アプリケーションレイヤーのフルフィルメントを実行するデザインに基づいて、実行計画を作成します。ユーザーは、特定のパッケージのデプロイメントのステータスを監視し、デプロイされたアプリケーションのグラフィック表現を表示できます。それには、コンポーネントレベルのプロパティとアクションも含まれています。

デザインの発行

デザインを発行すると、サービスコンシューマーに対するオファリングとして使用可能になります。デザインを発行するには、Hewlett Packard Enterprise CSAライセンスがインストールされている必要があります。

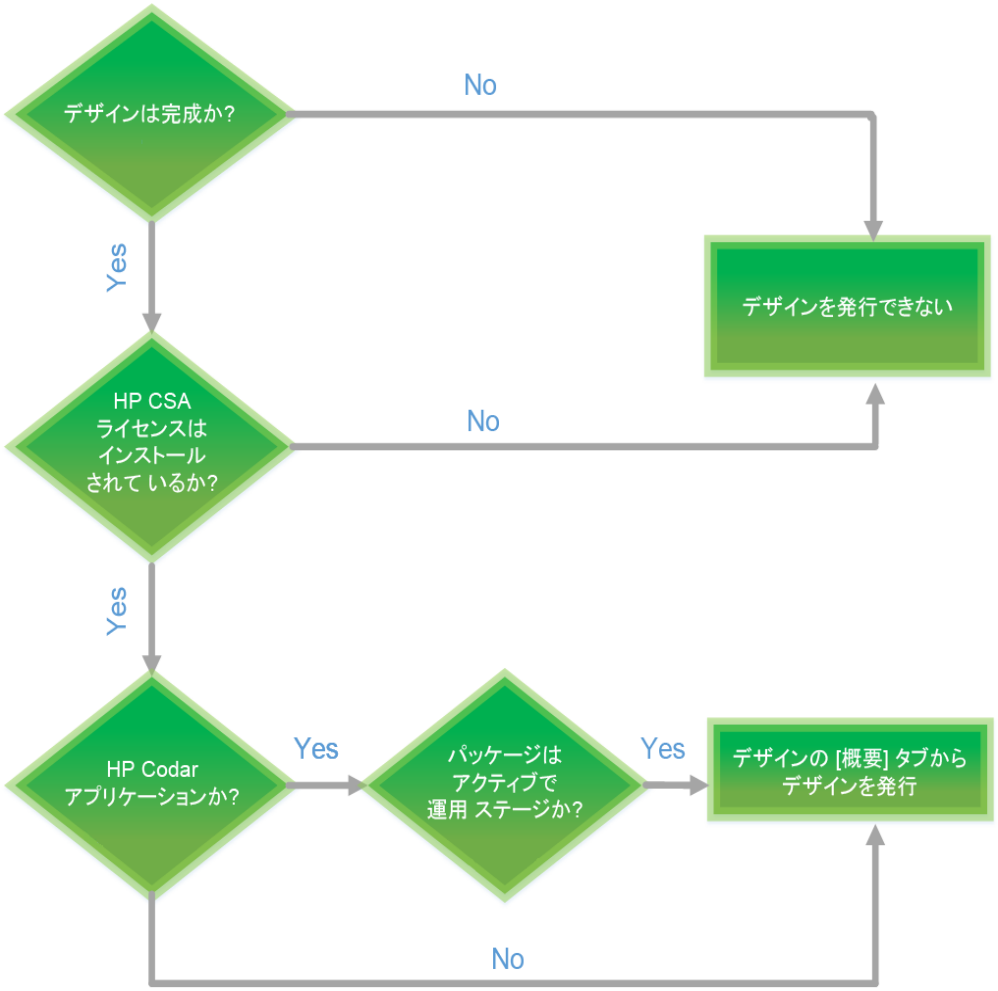
運用ステージのアクティブパッケージを持つ完全デザインは、パッケージ固有のプロパティをデザインの一部として持ち、発行可能です。

運用ステージのアクティブパッケージを持つ部分デザインは、パッケージ固有のプロパティをデザインの一部として持ちますが、運用パッケージをデプロイすることによって最終的な構成されたデザインを作成するまでは発行できません。

部分デザインの発行方法は、インストールされているライセンスによって異なります。

- 運用ステージまで進んだCodarアプリケーションデザインは、運用インフラストラクチャーにデプロイされ、構成された運用デザインがその後の運用デプロイメントで見えるようになります。その後、デザインをサービスコンシューマーに発行できます。
- Codarアプリケーションデザインでないデザインは、[テスト] タブから構成されたデザインとして保存する必要があります。その後、デザインをサービスコンシューマーに発行できます。

次の図は、使用するライセンスでデザインを発行できる条件を示しています。



ユースケース: カスタマイズ可能なリリースパイプライン

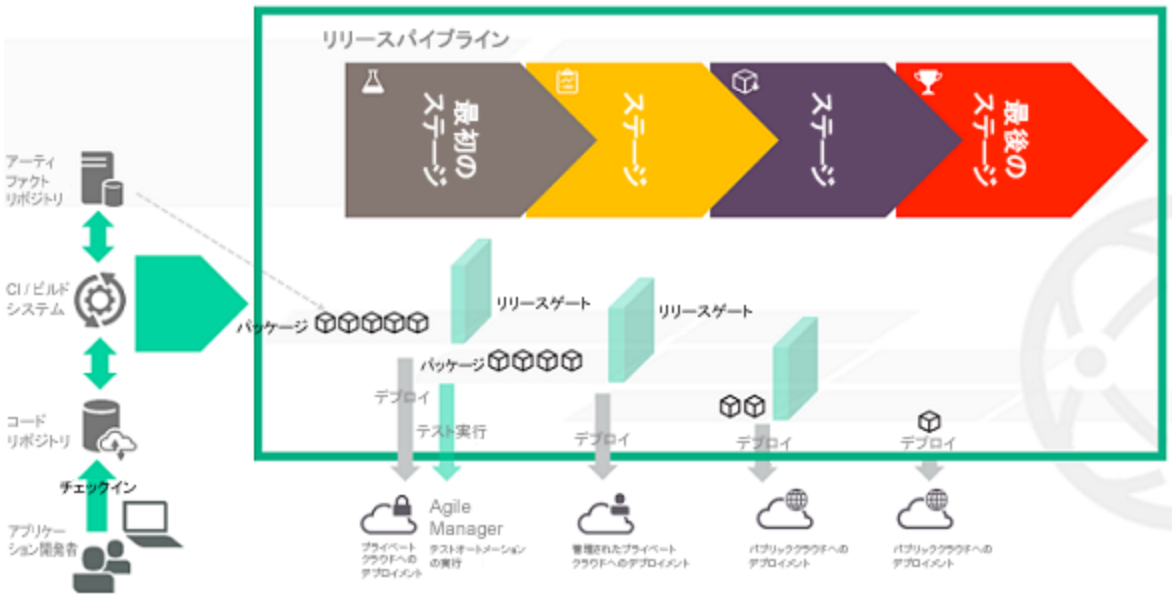
この目的は、ユーザーが独自のカスタムリリースパイプラインを構築し、アプリケーションチームごとに別々のライフサイクルステージを定義して使用できるようにすることです。次に、ユーザーが独自のリリースパイプラインを定義

する手順の概要を示します。これらの手順の実行方法については、Codarのオンラインヘルプを参照してください。

1. ロールを作成し、アクセス許可を追加する
2. ライフサイクルステージを作成し、各ライフサイクルステージにロールを関連付ける
3. ライフサイクルステージをアプリケーションデザインに追加する
4. リリースゲートアクションを各ライフサイクルステージに追加する
5. 連続昇格APIを使用してパッケージを作成する

すべてのリリースゲートが正常に実行されると、パッケージは最後のライフサイクルステージに移動します。

次の図は、カスタマイズ可能なリリースパイプラインを示しています。



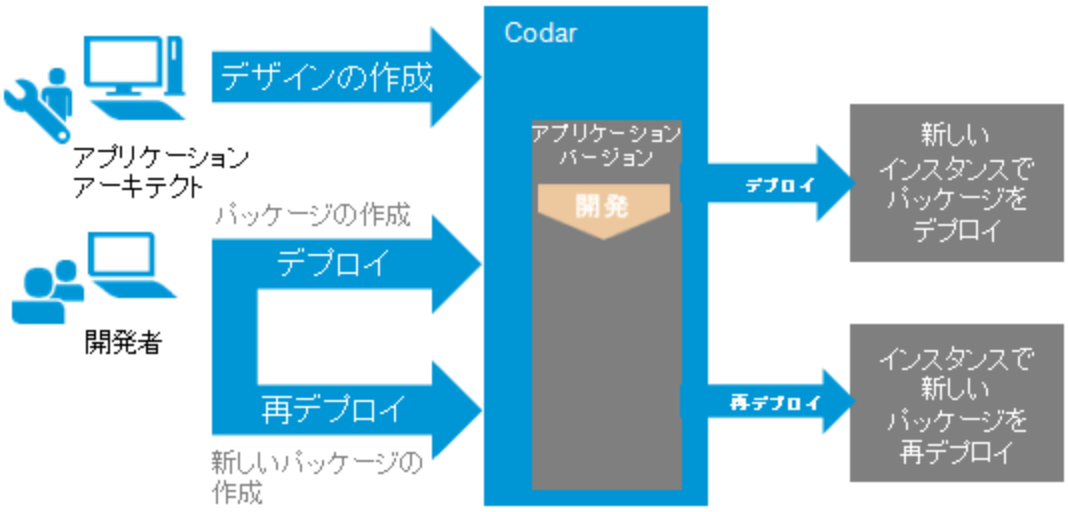
ユースケース: パッケージのデプロイと再 デプロイ

この目 的 は、パッケージをデプロイし、同じインスタンスを使 用して、古 いバージョンがデプロイされているインスタンス上 で新しいバージョンのパッケージを再 デプロイすることです。

アプリケーションアーキテクト が、アプリケーションをモデル化 し、パイプライン管 理 用 にマークを付 けます。次 に、開発 者 がパッケージを作 成 してデプロイします。パッケージがデプロイされると、新 しいインスタンスが作 成 されます。デプロイメント はトポロジデザインを基 にしています。

デプロイメント の後 に、同じインスタンスを使 用して、パッケージの新 しいバージョンを再 デプロイすることができます。インスタンスを新 しいパッケージまたはビルド にアップグレード したりパッチを適 用 したりすることができます。

次 の図 は、このプロセスを示しています。



ユースケース: デプロイメントとスケールアウト

目的は、スケーラブルスタックを作成し、デプロイメント後に必要に応じてスタックをスケールアウトすることです。アプリケーションアーキテクトが、アプリケーションをモデル化し、パイプライン管理用にアプリケーションにマークを付けます。アーキテクトが、さまざまなライフサイクルステージでスケールアウトする必要があるコンポーネントを識別します。スケーラブルスタックには、1つのコンポーネントまたはコンポーネントのグループを含めることができます。開発中に、スタックを1つスケールインしたり、テスト中にスタックを2つにスケールアウトしたりすることができます。

次のステップ

『Codar Installation Guide』および『CodarConfiguration Guide』に、このソフトウェアをダウンロードし、インストールして構成する方法の説明が記載されています。『CodarAPI and CLIReference』では、REST APIの概要が説明され、各APIの詳細を参照する方法が説明されています。さらに、コマンドラインインタフェースについても説明されています。また、アプリケーションからオンラインヘルプにアクセスし、具体的なタスクのヘルプ情報を取得することができます。