Codar

ソフトウェアバージョン: 1.70

コンセプトガイド

ドキュメントリリース日 : 2016年7月
ソフトウェアリリース日 : 2016年7月

Hewlett Packard Enterprise

ご注意

保証

Hewlett Packard Enterprise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、Hewlett Packard Enterprise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、Hewlett Packard Enterprise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。

OpenStack® Word MarkおよびSquare O Designは、単独でも両方でも、OpenStack Foundationの米国およびその他の国における登録商標であり、OpenStack Foundationの許可を得て使用されています。

OracleおよびJavaは、Oracle Corporationおよびその関連会社の登録商標です。

RED HAT READY™ロゴとRED HAT CERTIFIED PARTNER™ロゴは、Red Hat, Incの商標です。

この製品には、'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。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.hpe.com/

このサイトを利用するには、HP Passportに登録してサインインする必要があります。HP Passport IDに登録するには、ソフトウェアサポートサイトで [**Register**] をクリックするか、HP Passportログインページで [**Create an Account**] 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、営業担当にお問い合わせください。

サポート

ソフトウェアサポートサイトを参照してください。https://softwaresupport.hpe.com

Hewlett Packard Enterprise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
- HPE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、HP 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。HP Passport IDに登録するには、サポートサイトで [**Register**] をクリックするか、HP Passportログインページで [**Create an Account**] をクリックします。

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.hpe.com/web/softwaresupport/access-levels

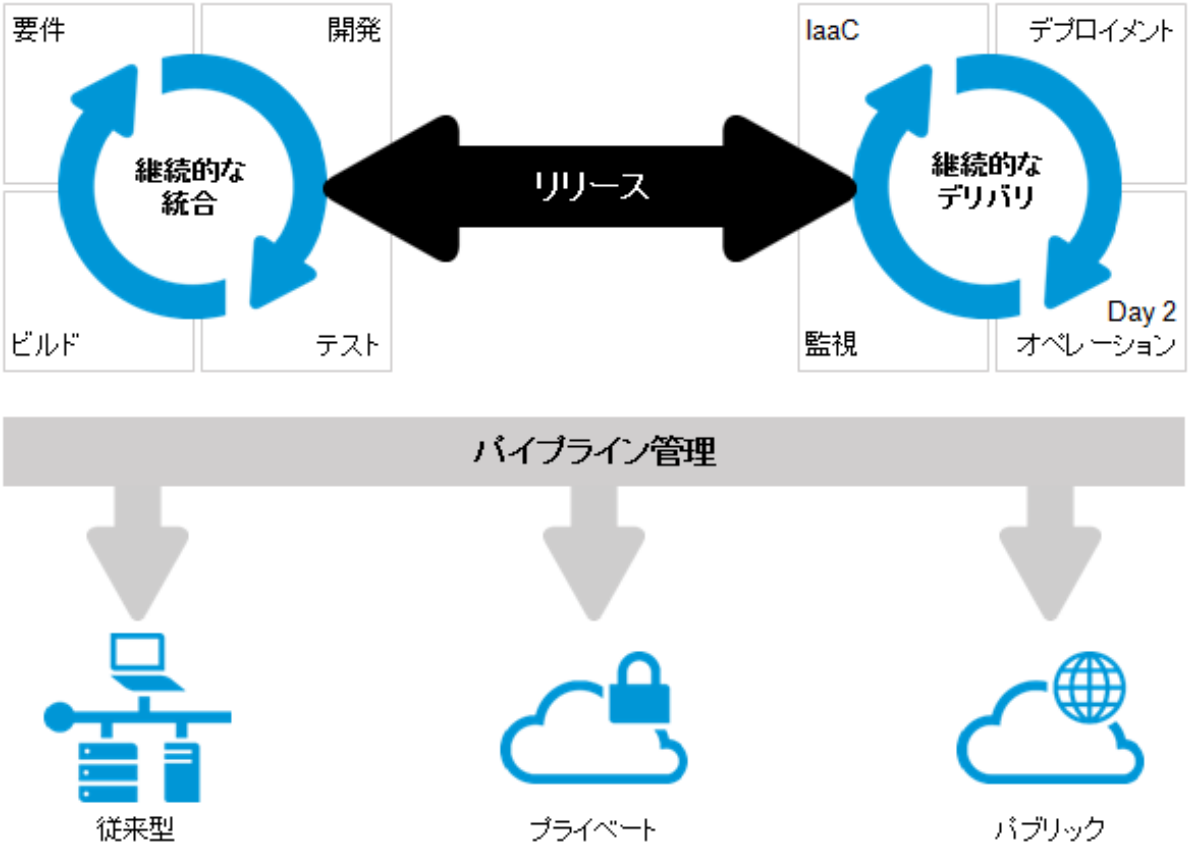
目次

Codar

組織が連続統合を連続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。

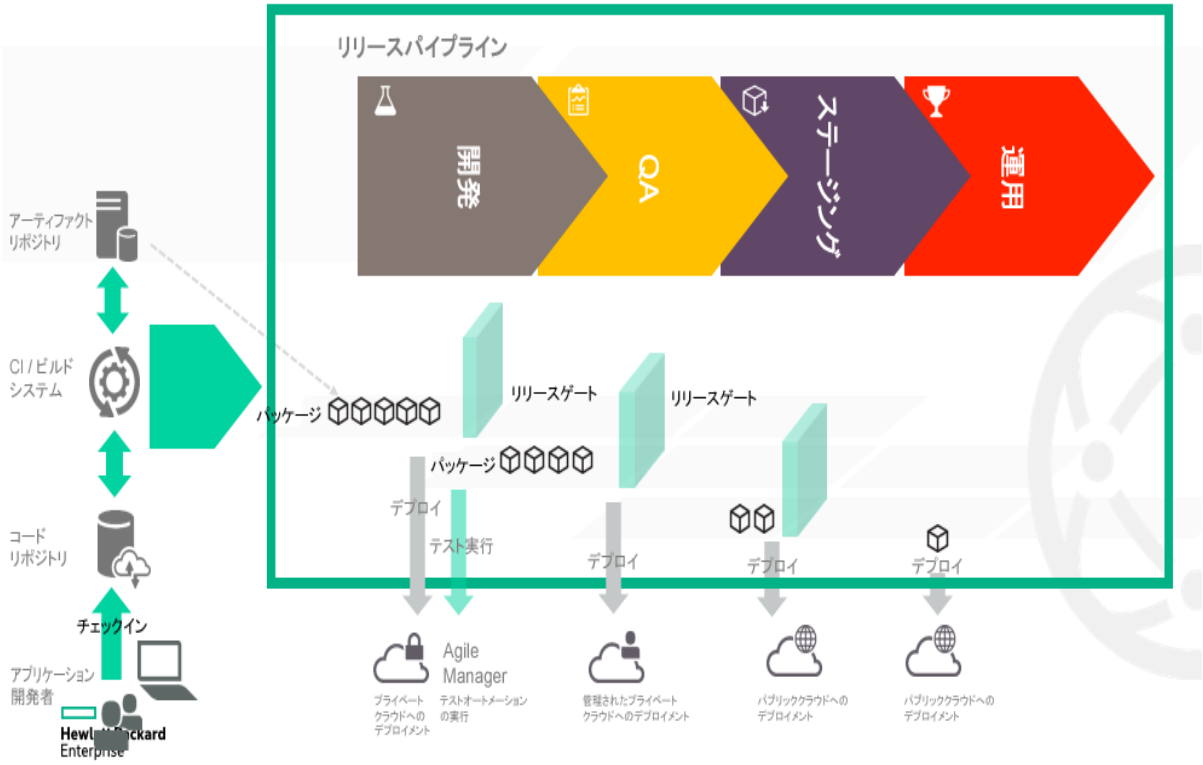
DevOpsは、コラボレーション、自動化、ガバナンスに関する一連の方針、方法、手法を使用して、開発 (Dev) 環境と運用 (Ops) 環境の溝を埋めるためのフレームワークを提供します。この目的は、ビルドまたはアセンブリの連続統合を、異種環境間で、再現性があり一貫性のあるアプリケーションデプロイメントに広げることです。

次の図は、DevOps環境での連続統合と連続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 な連 続 デリバリを促 進 します。Codarは、連 続 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。連 続 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。



コンポーネント

コンポーネントはサービスデザインの要 素 です。[コンポーネント] タブには、トポロジコンポーネントのみが表 示 されます。シーケンスコンポーネントは、プロバイダーやプロバイダータイプには関 連 付 けられません。[コンポーネント] タブでは、特 定 のプロバイダーインスタンスに関 連 付 けられているトポロジコンポーネントの表 示 と、トポロジコンポーネントの管 理 を行 うことができます。

サービスデザイン

サービスデザインを作成、構成、変更して、オンデマンドの自動サービスデリバリを実現します。サービスデザインは、クラウドの自動化のためのレシピであり、再利用可能なサービスコンポーネントが含まれています。サービスデザインに含まれるサービスコンポーネントとそれらの間の関係は、サービスを作成するためのフレームワークを定義します。

サービスデザインは、コンシューマーがサービスをオーダーするときに選択するオプションの構造も指定します。デザインを複数のサービスオファリングに再利用し、各サービスオファリングをコンシューマー組織とグループの個々のニーズに合わせてカスタマイズすることができます。また、Codarに付属するサービスデザインを利用したり、複数のCodar間でデザインをエクスポート/インポートしたりすることもできます。

シーケンスデザインとトポロジデザインを作成できます。

シーケンスデザイン

シーケンスデザインは、サービスコンポーネントライフサイクルの方向付けられた実行を指定するもので、各コンポーネントのデプロイ時に選択されるリソースを制御する仕組みを提供します。シーケンスデザインを作成する際には、サービスコンポーネントにリソースオファリングを関連付けることにより、選択できるプロバイダーを制限します。この関連付けまたはリンクにより、サービスコンポーネントをデプロイすると、リソースオファリングがプロビジョニングされます。コンポーネントテンプレートでリソースオファリングを関連付けることもできます。

シーケンスデザインは、複雑なサービスやランブックオートメーションを必要とするサービスに使用します。例としては、旧式のデータセンターシステムとの統合などが挙げられます。シーケンスデザインは、ライフサイクル実行を定義する方向付けられたコンポーネント階層として作成します。シーケンスデザインでは、コンポーネントを使用して、複数のオートメーションプロバイダーを1つのエンティティ内にグループ化します。また、ライフサイクルアクションの明示的な指定が可能です。

トポロジデザイン

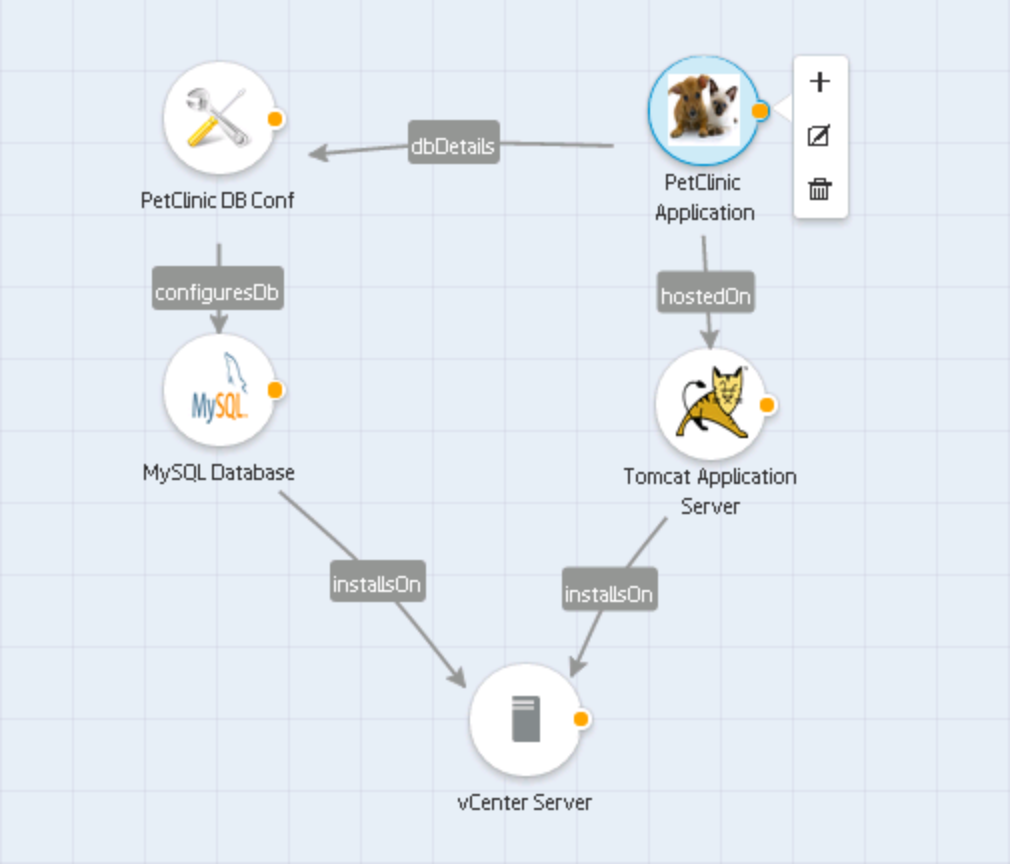
トポロジデザインは、コンポーネント、関係、プロパティを指定します。シーケンスデザインがプロビジョニングの順序と実行されるアクションのシーケンスを明示的に定義するのに対し、トポロジデザインは宣言的性質を持ち、明示的なアクションやシーケンスを含みません。トポロジデザインでは、プロビジョニングシーケンスはコンポーネントの間に存在する関係から推測されます。

トポロジデザインはChef、Puppet、Server Automation、Operations Orchestrationフローベースのコンポーネントを使用して実現されるIaaS (Infrastructure as a Service)、PaaS (Platform as a Service)、およびSaaS (Software as a Service) デプロイメントに対して使用します。

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のデプロイメントを自動化する場合、ユーザーはアプリケーションデプロイメントの終了状態 (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)。



トポロジの構 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。

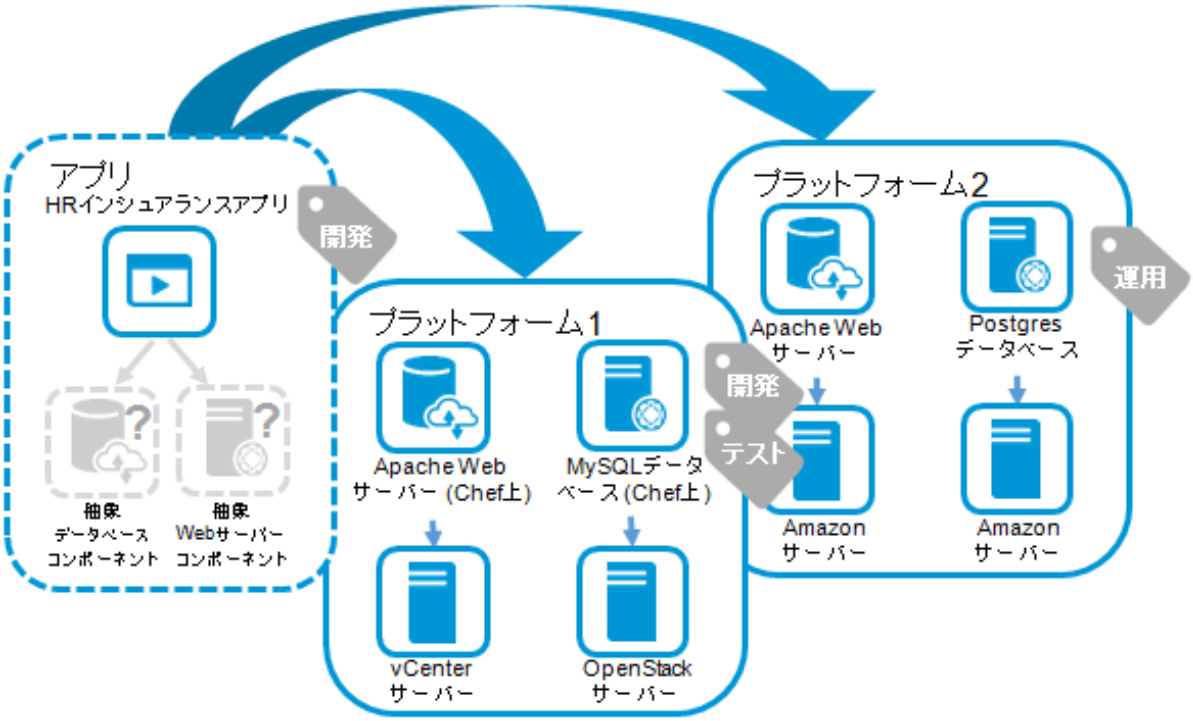
アプリケーションデザインには、次 の2つのタイプがあります。

- 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。



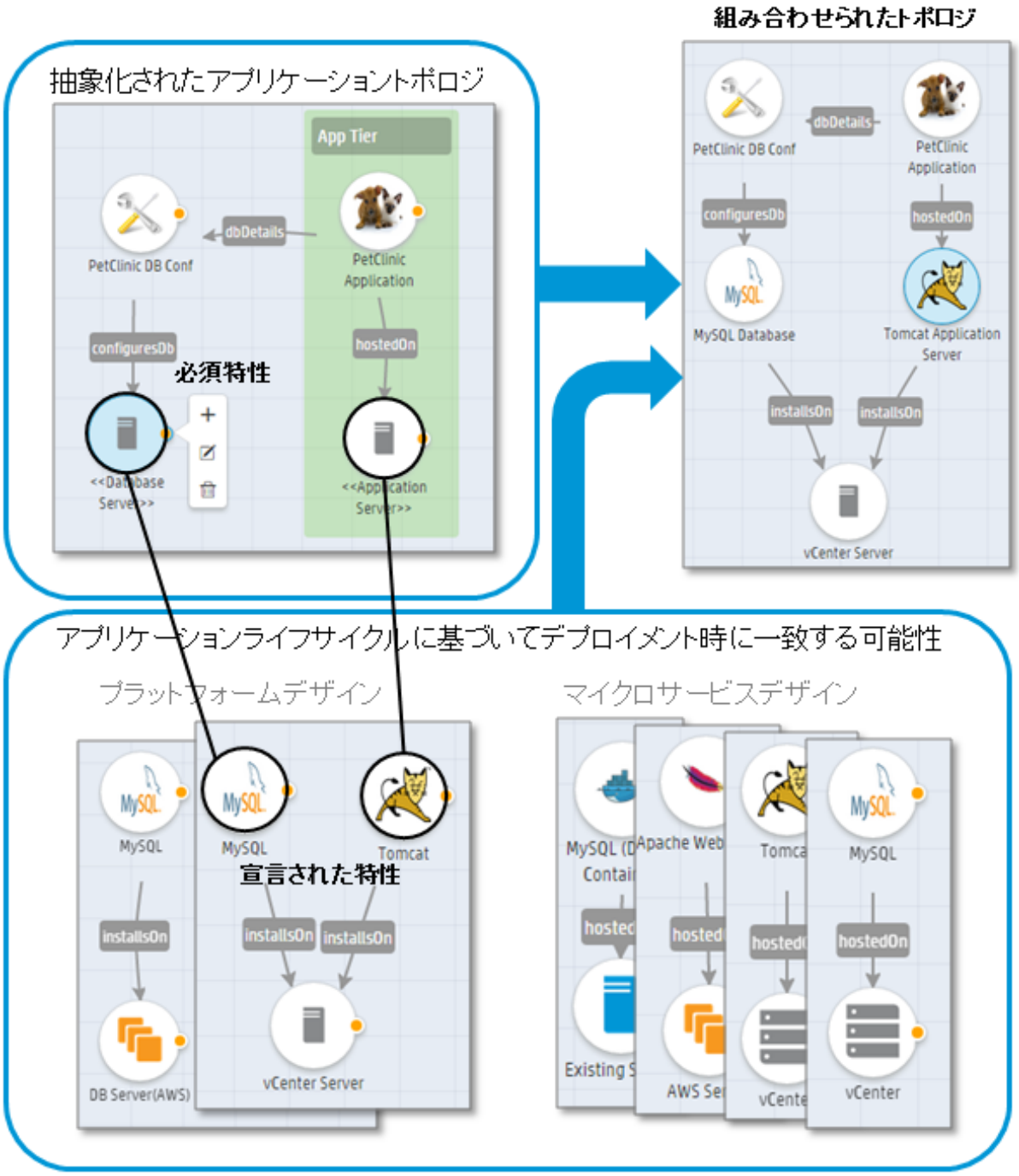
マイクロサービス

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

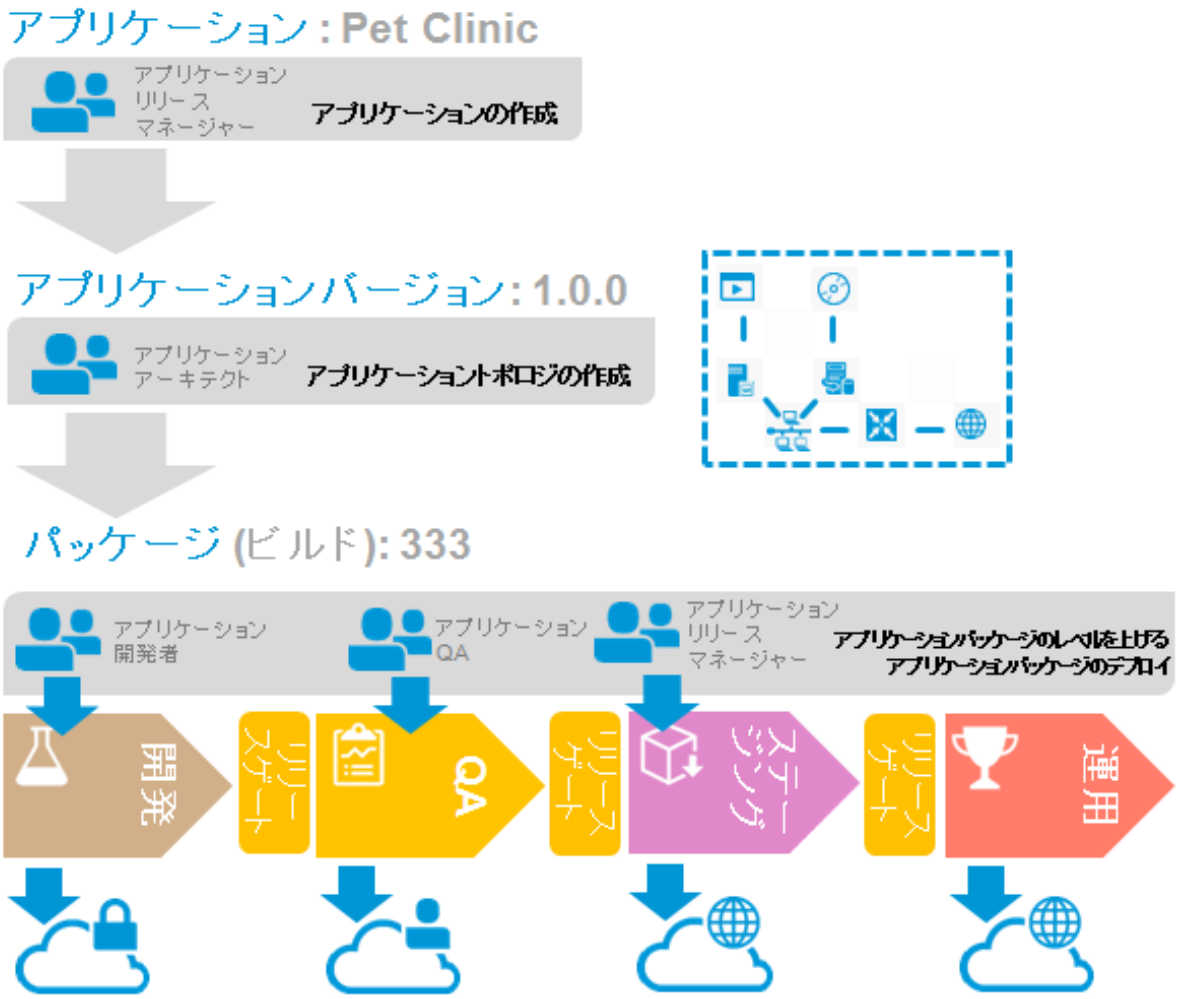
アプリケーションのデプロイメントの自 動 化 は、複 雑 で時 間 を要 するプロセスであり、多 くの調 査 を必 要 とします。開 発 と運 用 でアプリケーションのデプロイ方 法 が異 なると、多 くのエラーの原 因 になります。アプリ

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 、テスト、ステージング、運 用 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。

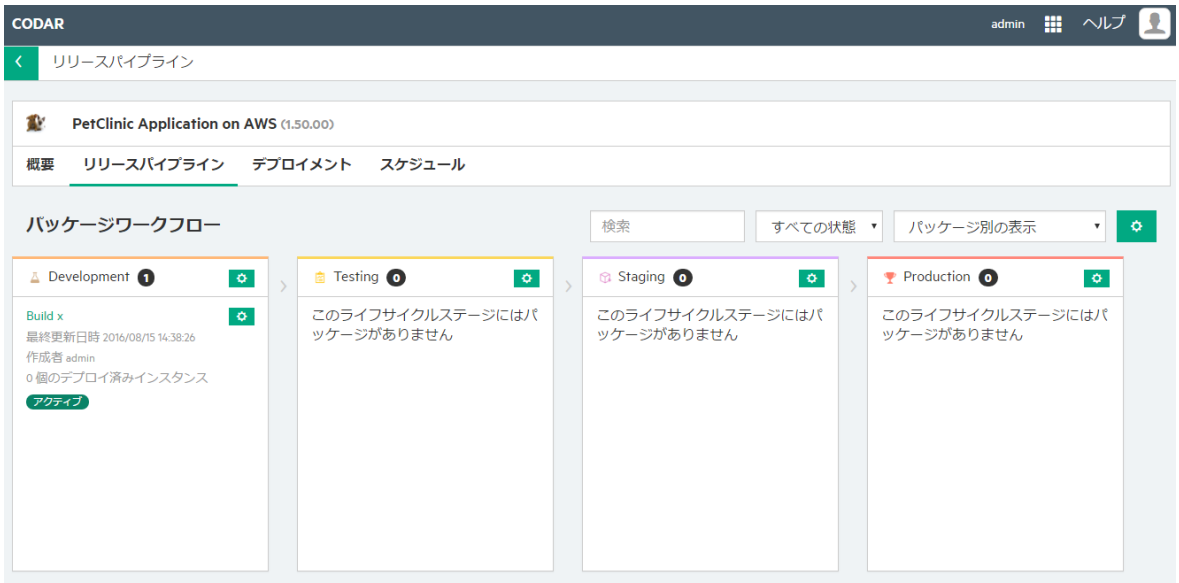
> **注：** インスタンスが関 連 付 けられているパッケージは削 除 できません。

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。

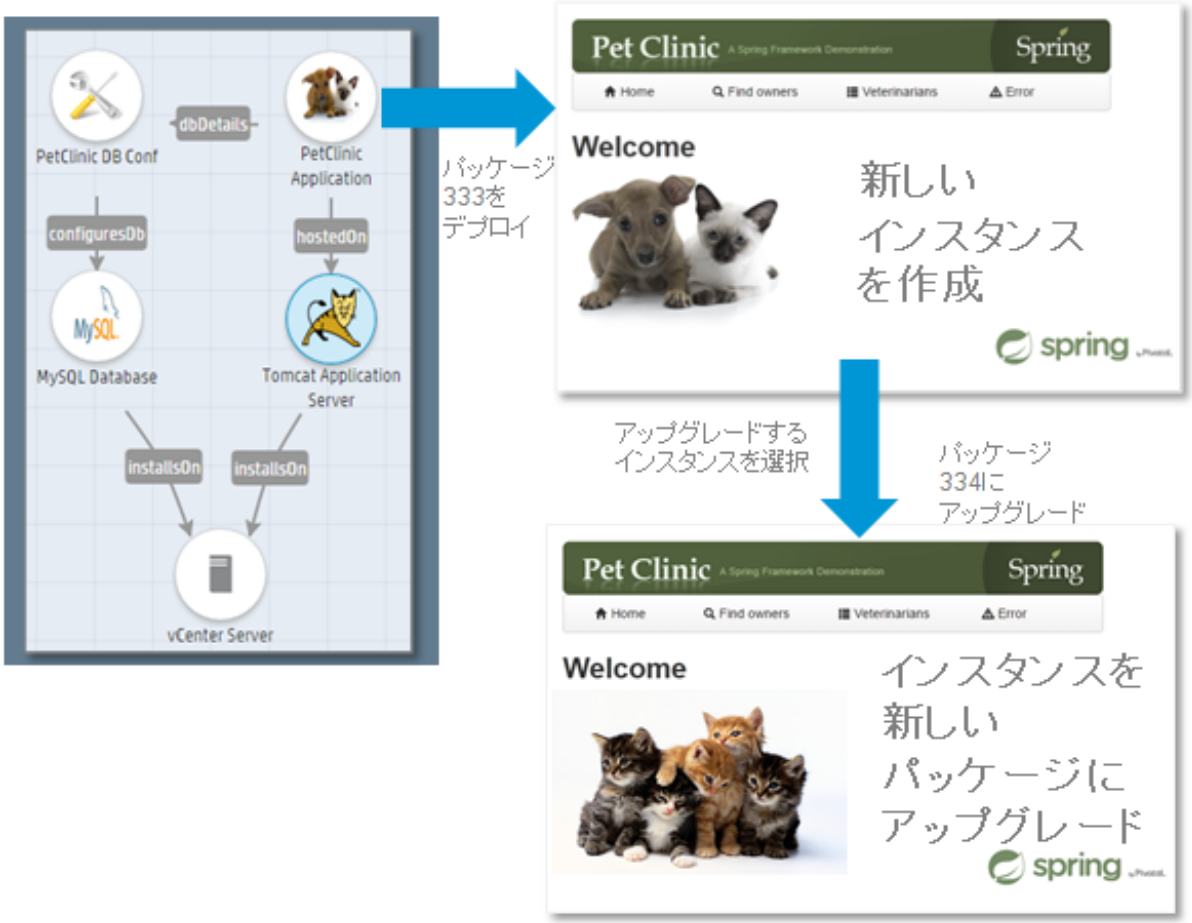
パッケージを作 成 してデプロイすると、新 しい仮 想 マシンが作 成 され、パッケージがデプロイされます。デプロイ済 みのインスタンスについてテストを実 行 し、パッケージのレベルを上 げるか拒 否 することができます。

次 の画 像 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。

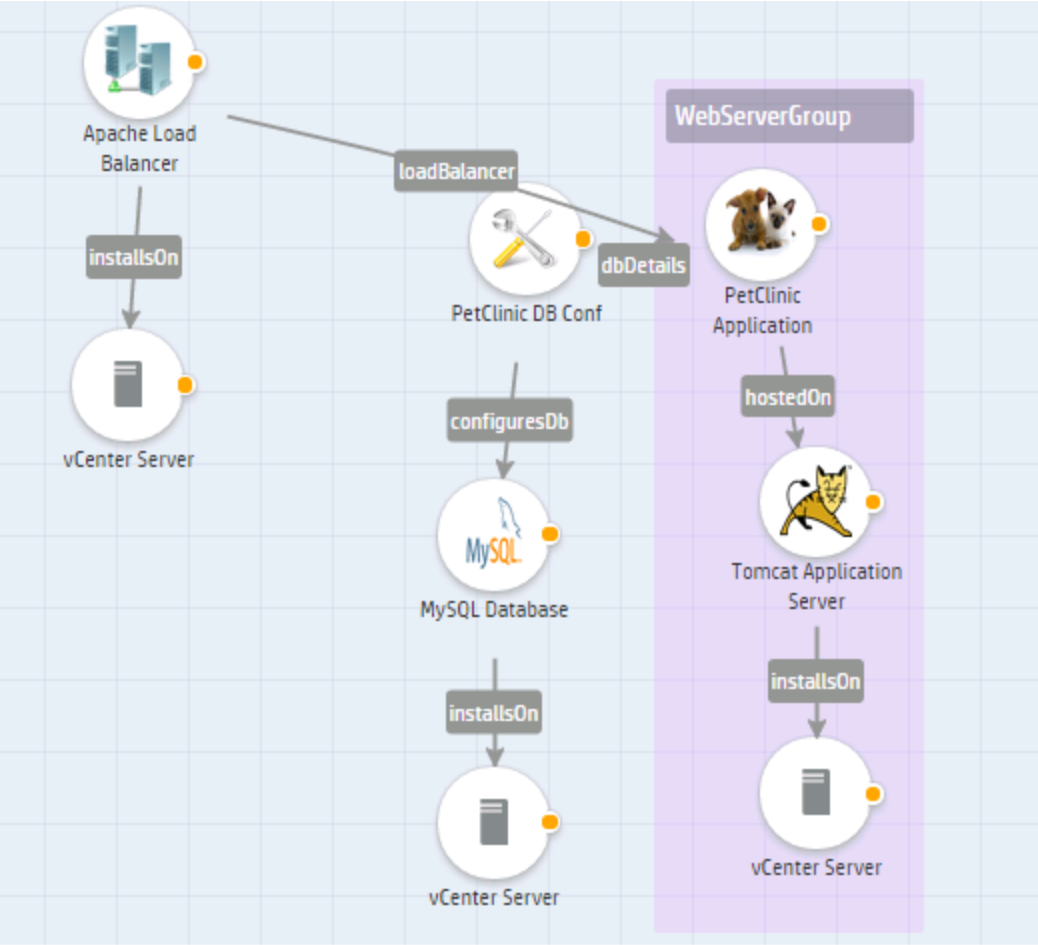


スケールアウト

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。

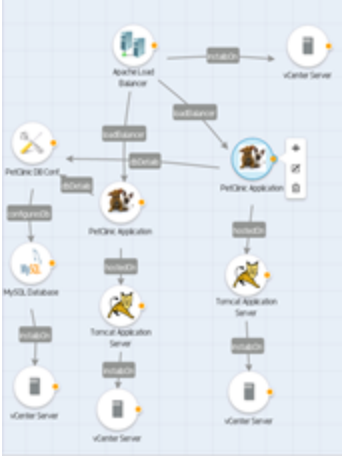
デプロイメント が完 了 した後 でスケールアウト することができます。スケールアウト すると、スタック全 体 が複 製 されます。

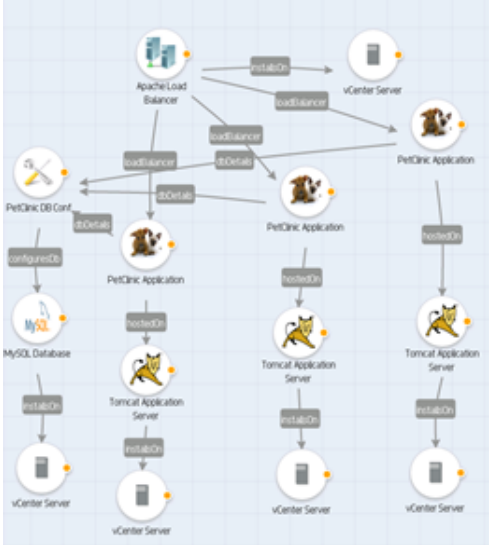
たとえば、下 の図 は、webServerGroupという名 前 の論 理 的 なグループとしてのWeb層 を示 しています。



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

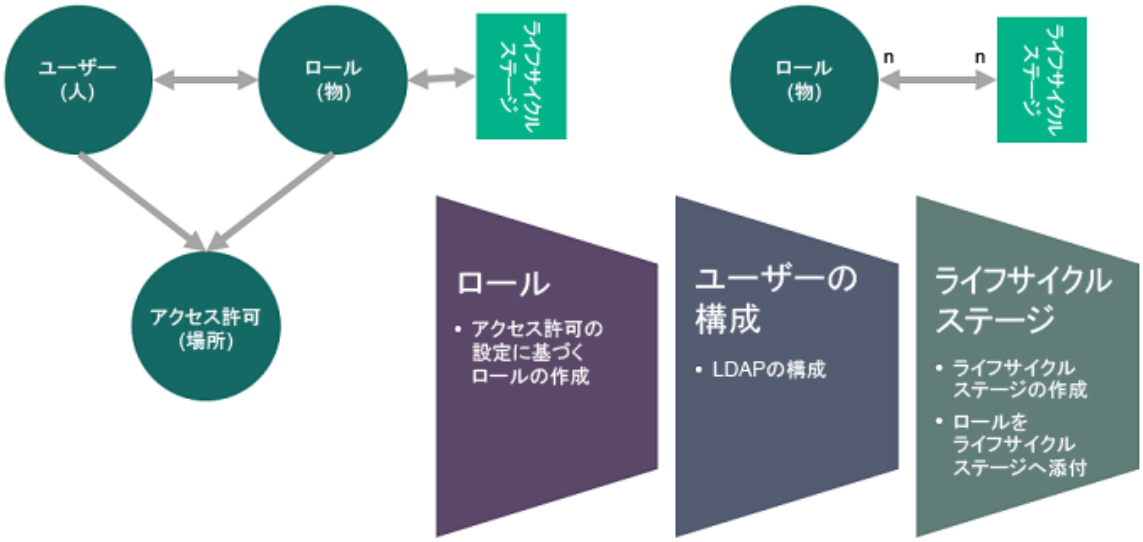




ロールとユーザーアクセス

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。

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: 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: 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: 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: 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)	Yes	Yes	Yes	Yes	Yes
中間	Yes	Yes	Yes	Yes	Yes
最終ステージ (通常は運用ステージ)	×	Yes	Yes	Yes	Yes

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**: パッケージを削除します。パッケージはシステムから恒久的に削除されます。

> **注:** 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。

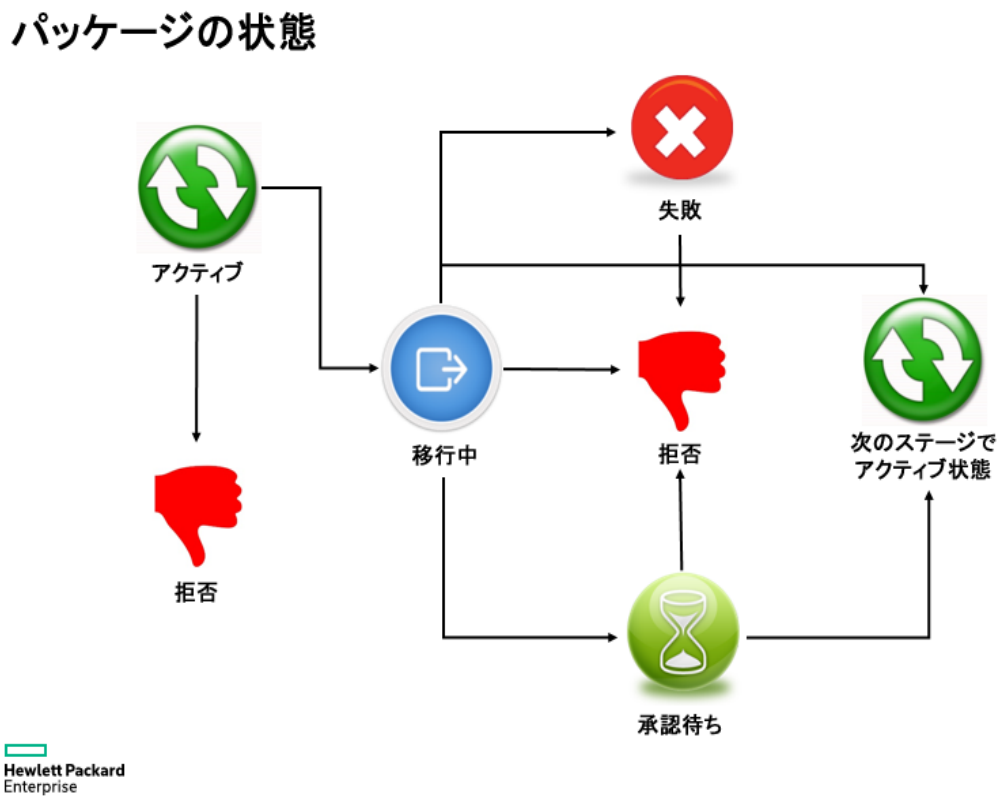
パッケージの状態

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サービス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、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サービス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サービス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4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

  Operations Orchestrationで指定したフローを実行します。一般的に、このアクションは、デプロイメントインスタンスを使用または使用せずに、特定のテストを実行する目的で使用できます。

  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。

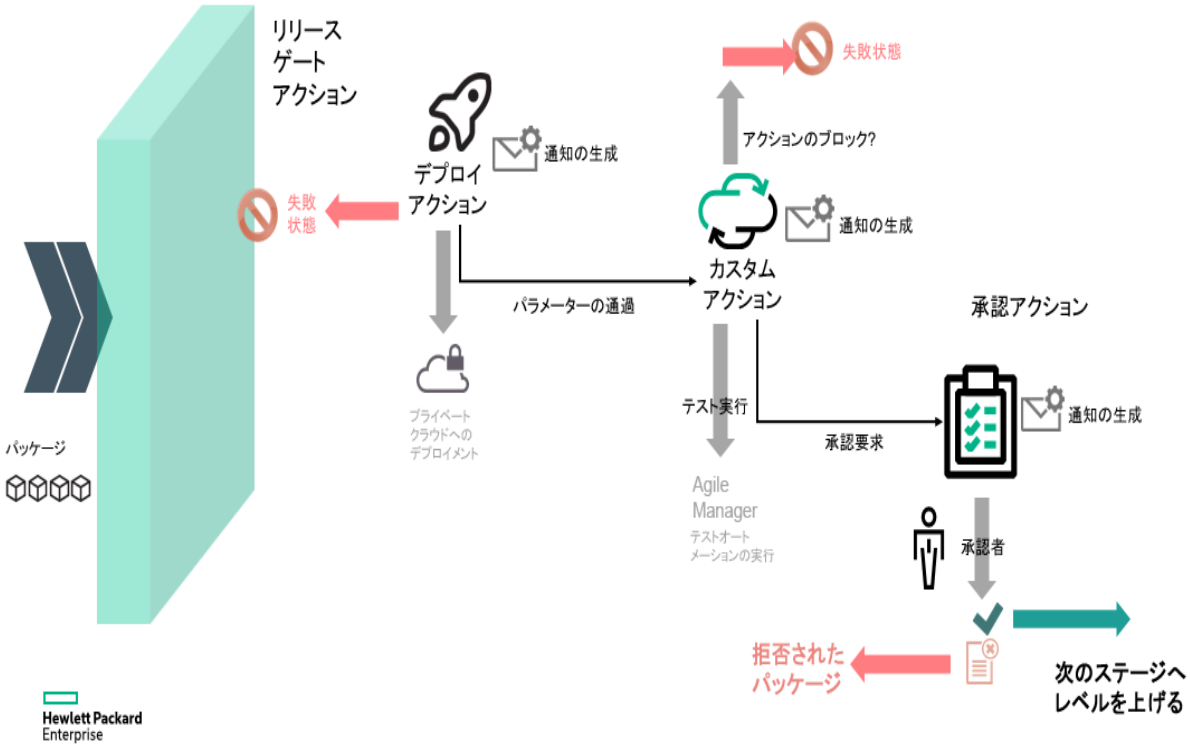
  承認アクションの作成、編集、削除については、Codarのオンラインヘルプを参照してください。

- テストセットアクション

  このアクションは、デプロイメントインスタンスを使用または使用せずに、Application Lifecycle Managementで特定のテストを実行する目的で使用できます。テストセットアクションとともに、UAT環境を追加することもできます。

  テストセットアクションの作成、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。

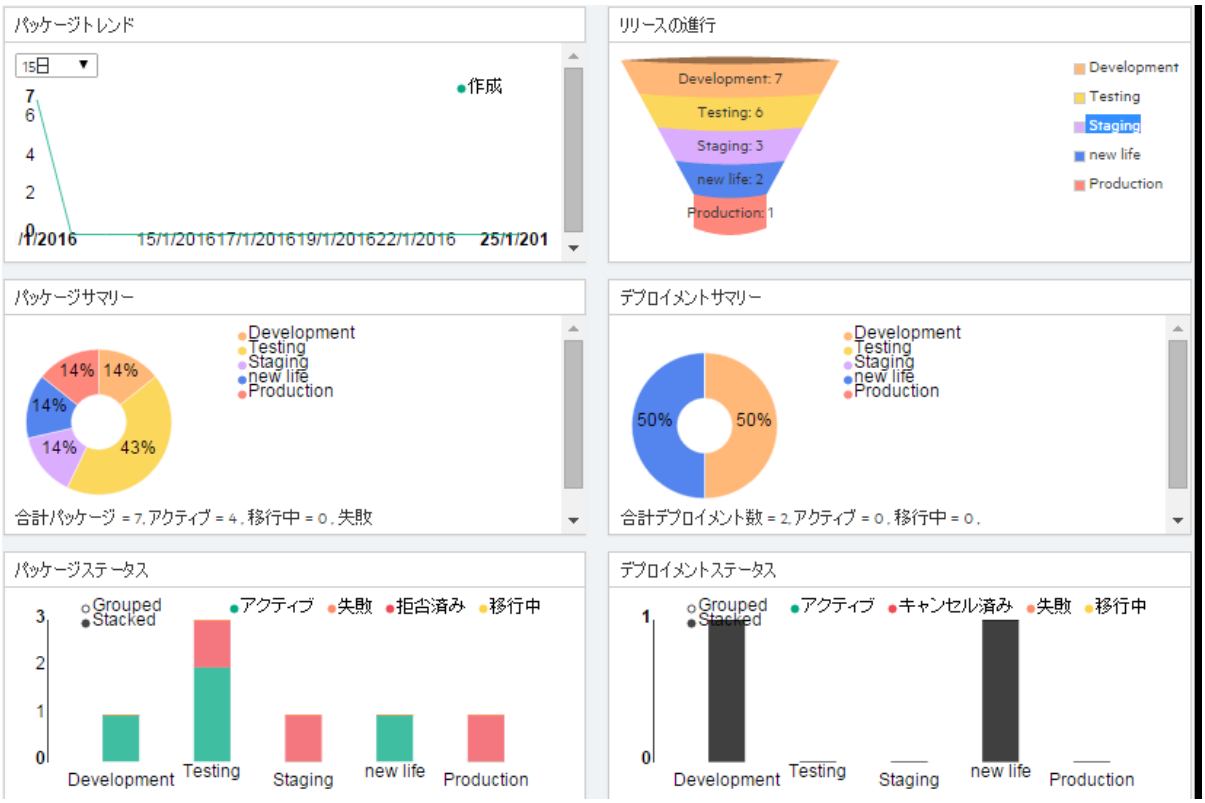


パイプライン統計

[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

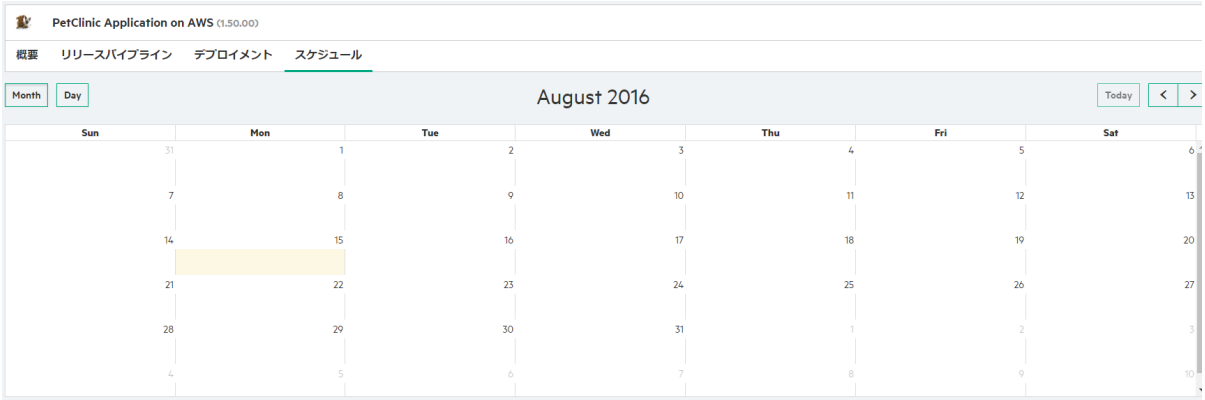
各ライフサイクルステージに合わせてアプリケーションレベルで異なる環境を選択できます。たとえば、デプロイメント用にvCenterを構成し、ステージング用にはパブリッククラウド環境を選択することができます。

リリースのスケジュール

リリースパイプラインを通じたパッケージの移行をスケジュールし、管理することができます。次のスケジュールをセットアップできます。

- 1つのライフサイクルステージから別のステージへのパッケージの自動昇格
- パッケージとデプロイメントの定期的な自動消去

スケジュールをカレンダーに表示して、ライフサイクルのステージ間のパッケージ昇格のスケジュールを視覚的に確認できます。

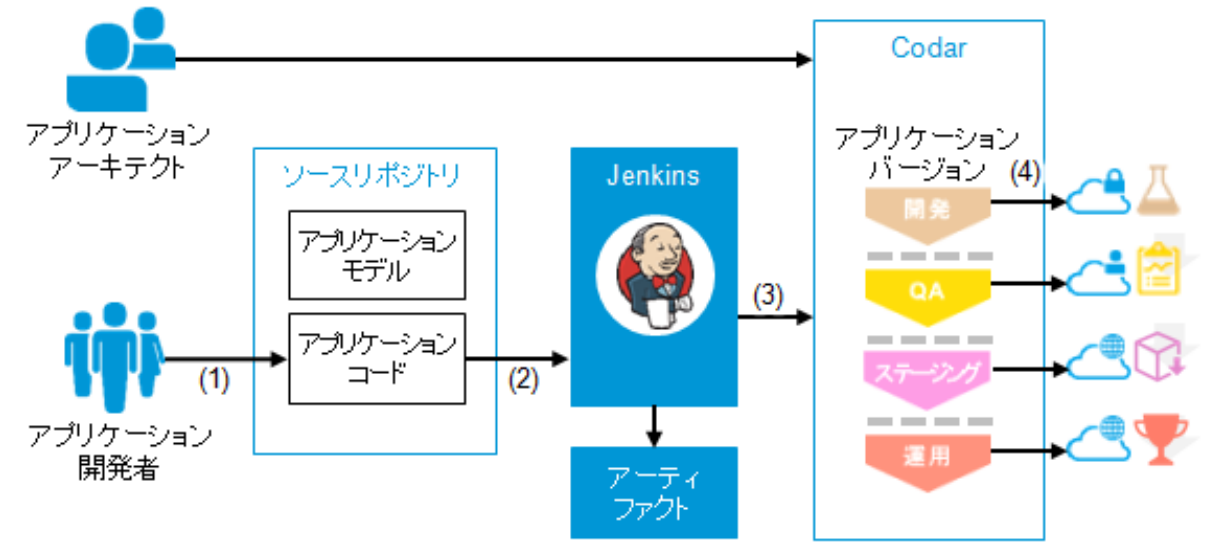


外部統合

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、連続的な統合、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、連続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。

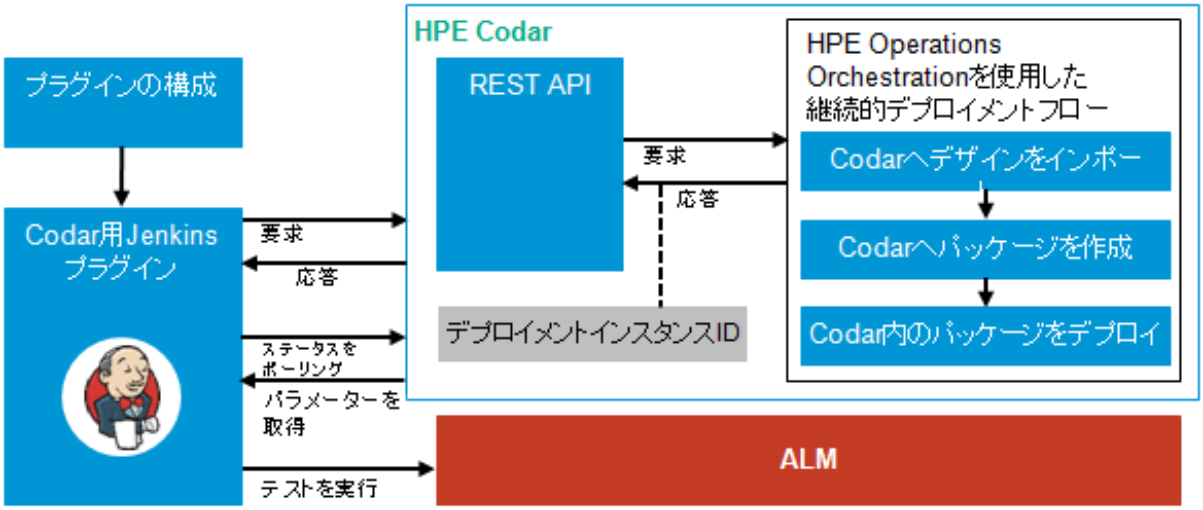


1. 開発者が変更をチェックインします。
2. 連続統合によってビルドが開始されます。
3. Jenkinsプラグインがパッケージを作成してデプロイします。
4. ライフサイクルステージでアプリケーションが異なる環境にデプロイされます。
5. 連続昇格の場合、すべてのリリースゲートアクションが正常に実行されると、パッケージは最終ライフサイクルステージに移動します。

> **注：** 同様のOOTB統合が、BambooおよびTFSに対しても利用できます。

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。

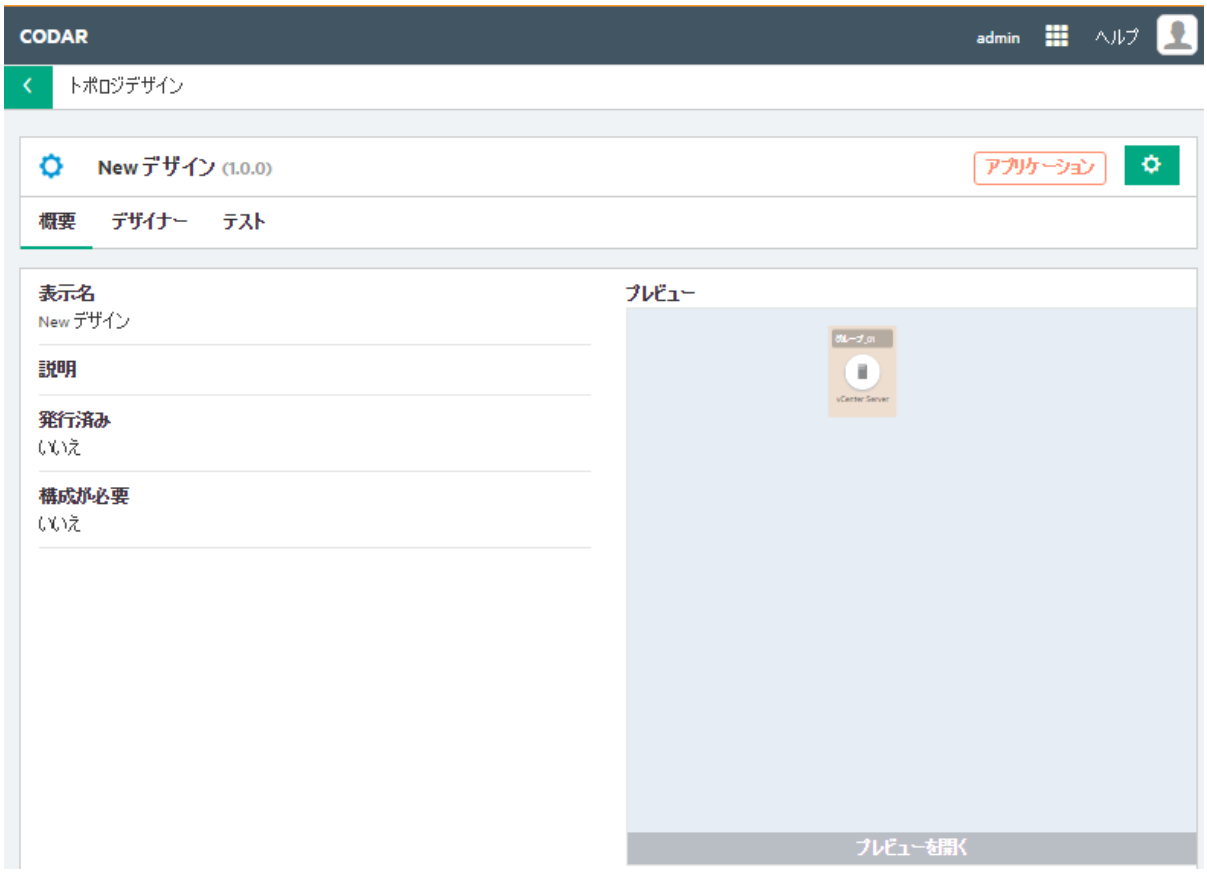


Infrastructure as code (IaaC)

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、テキストエディター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。



Dockerクラスターへのコンテナーのデプロイ

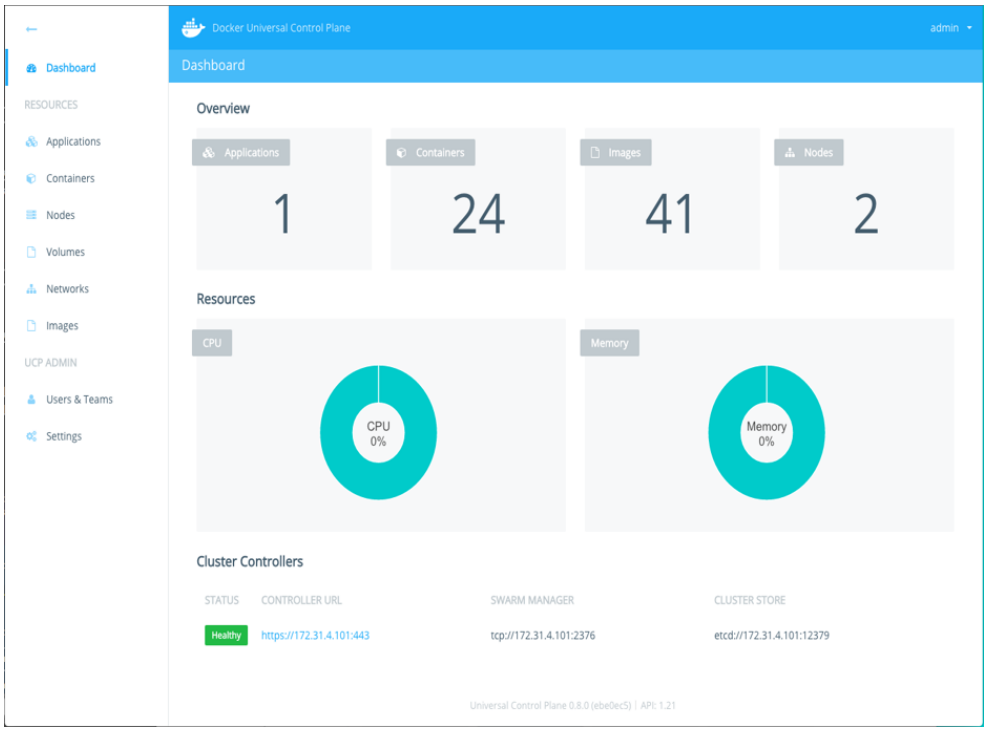
Docker Universal Control Planeには、Dockerアプリケーション用のオンプレミスまたは仮想プライベートクラウド (VPC) コンテナー管理ソリューションが用意されています。

Docker UCPでは、次の作業を実行できます。

- Docker UCPをCodarのリソースプロバイダーとして構成できます。
- Dockerコンテンツに基づいてデザインを作成できます。
- Docker UCPにコンテナーをデプロイし、パイプライン管理を実行できます。

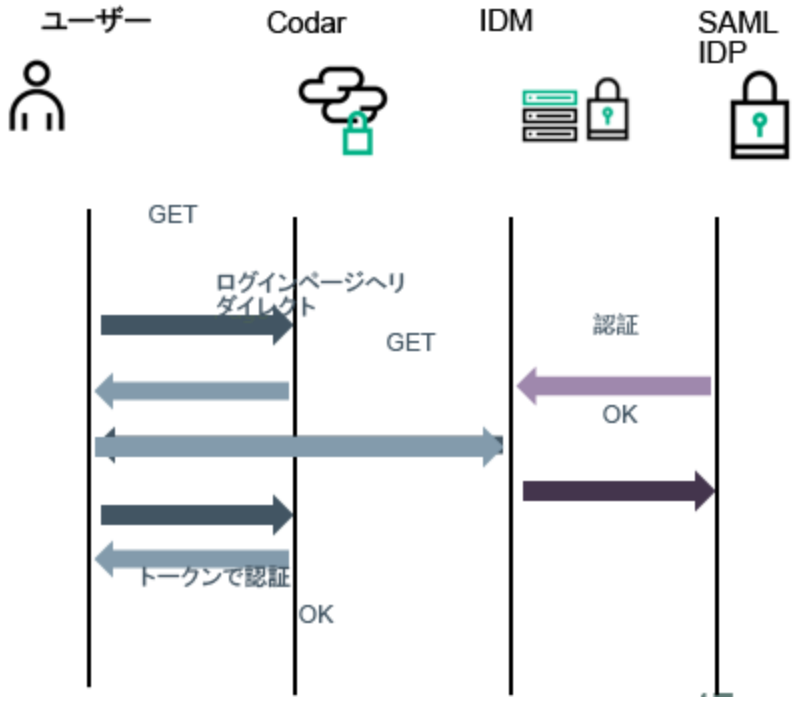
Docker Universal Control Plane

Docker UCPダッシュボード



CodarでのSAMLサポート

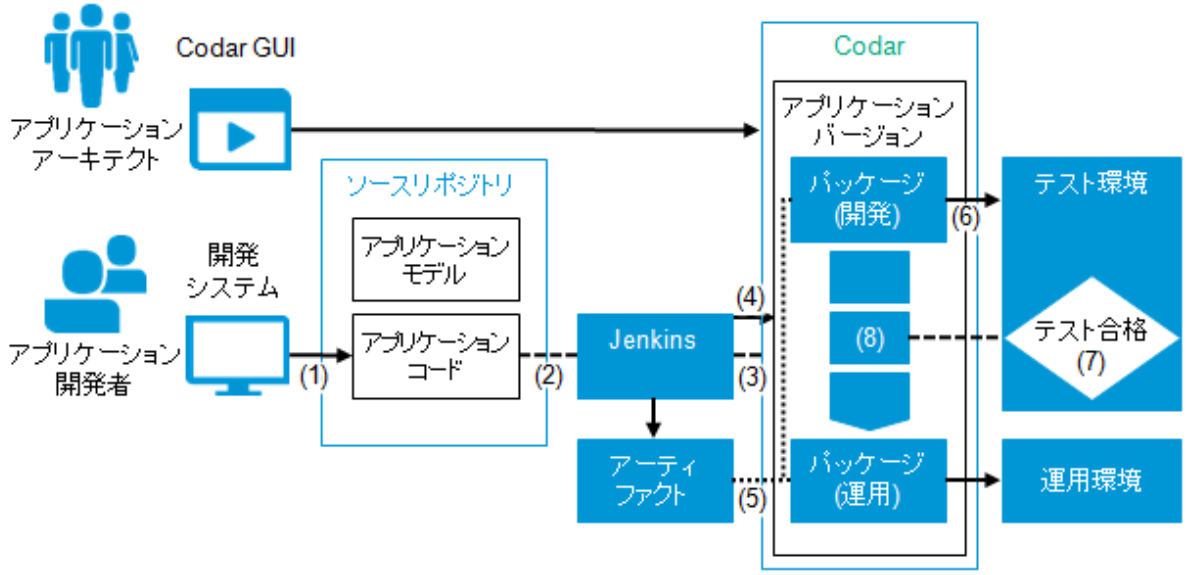
Codarでは、SAMLによるSSOを提供しています。1回だけ認証を(SAMLをサポートするアイデンティティプロバイダーを通じて)行えば、以後パスワードを入力せずにCloud Service Automationを使用することができます。


ユーザー
Codar
IDM
SAML
IDP
GET
ログインページへリ
ダイレクト
GET
認証
OK
トークンで認証
OK

ユースケース: 連続的な統合、デプロイメント、デリバリ

目的は、アプリケーションを連続統合 (CI) と連続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、連続デプロイメントプロセスが連続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などの各種デプロイメント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。

連続統合と連続デプロイメント

連続統合では、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、連続デプロイメントと連続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、連続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]/[シーケンス] 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、連続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、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。

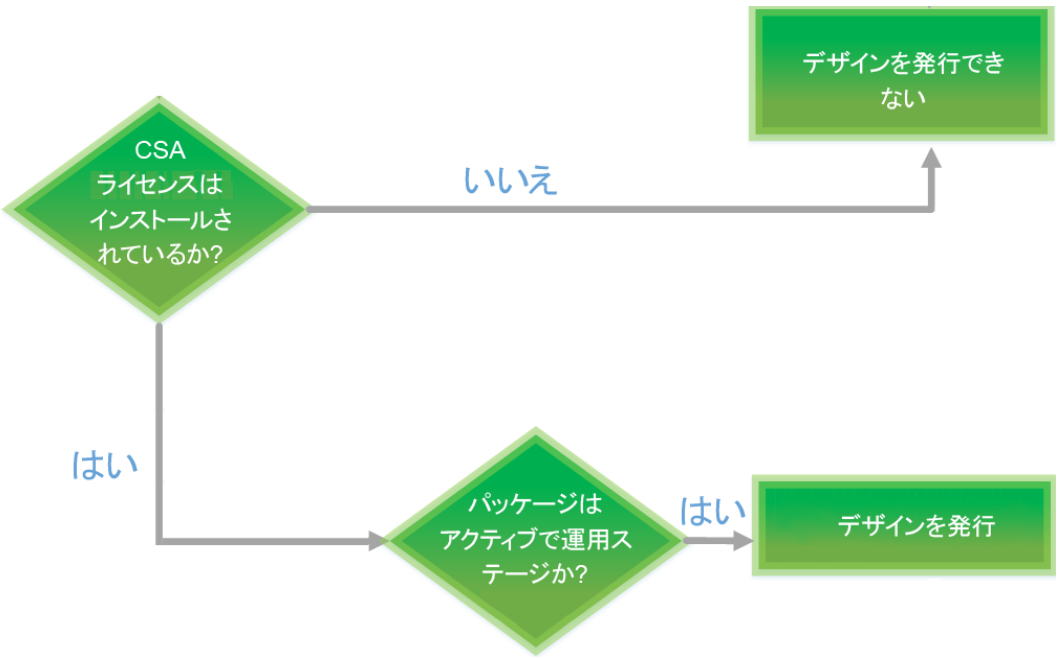
運用ステージのアクティブパッケージを持つ部分デザインは、パッケージ固有のプロパティをデザインの一部として持ちますが、運用パッケージをデプロイすることによって最終的な構成されたデザインを作成するまでは発行できません。

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。

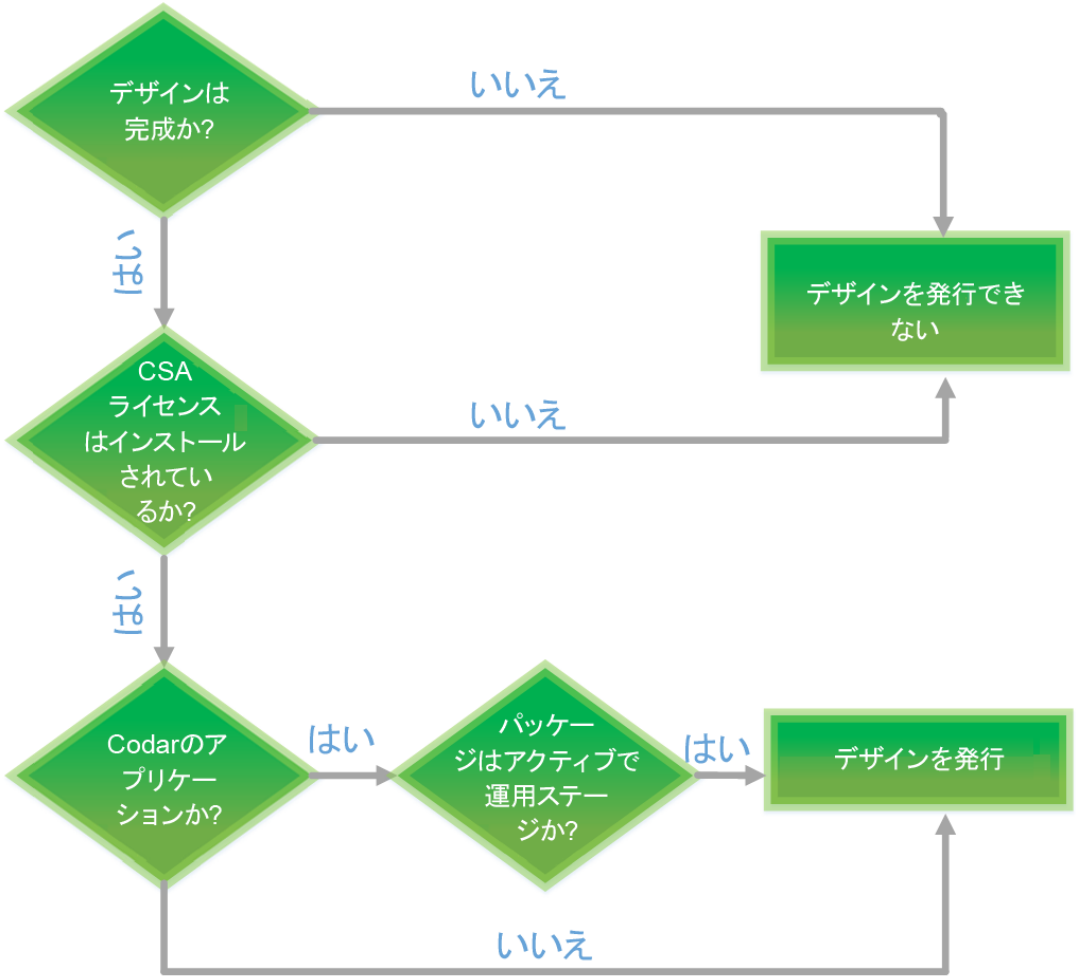
- 運用ステージまで進んだCodarアプリケーションデザインは、運用インフラストラクチャーにデプロイされ、構成された運用デザインがその後の運用デプロイメントで見えるようになります。その後、デザインをサービスコンシューマーに発行できます。
- Codarアプリケーションデザインでないデザインは、[テスト] タブから構成されたデザインとして保存する必要があります。その後、デザインをサービスコンシューマーに発行できます。

> **注：** シーケンスデザインは、パッケージがない場合、またはパッケージがパイプライン管理の最終ステージにあり、デプロイメントインスタンスがアクティブな場合にのみ発行できます。
>
> トポロジデザインがアプリケーションにタグ付けされている場合は、タグ付けを解除しないとトポロジデザインは発行できません。

次の図は、使用するライセンスでシーケンスデザインを発行できる条件を示しています。



次の図は、使用するライセンスでトポロジ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

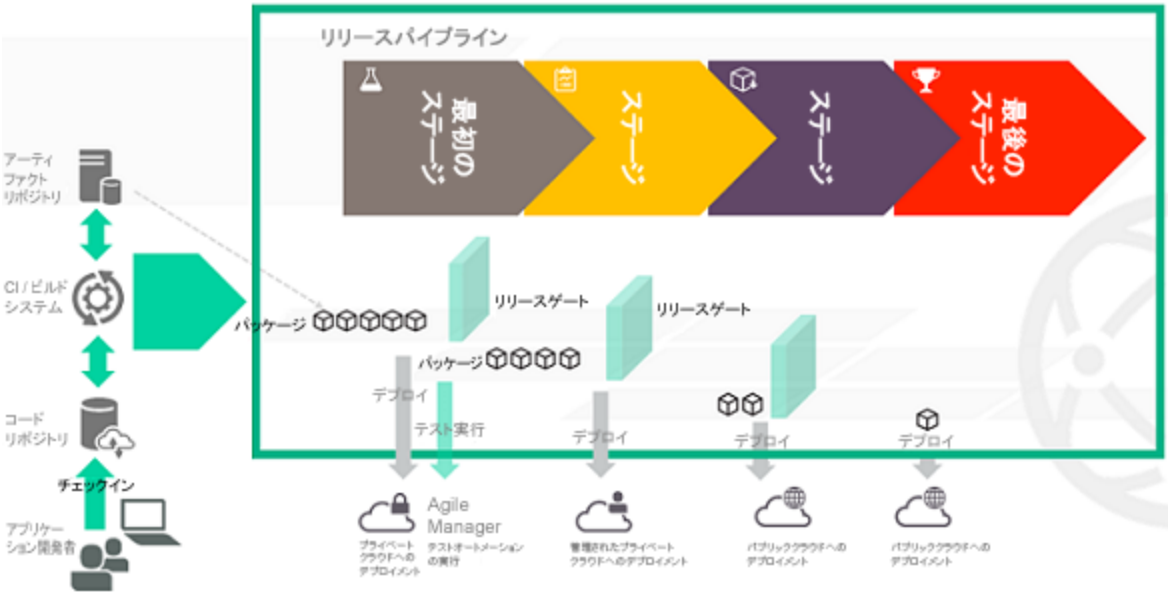
この目的は、ユーザーが独自のカスタムリリースパイプラインを構築し、アプリケーションチームごとに別々のライフサイクルステージを定義して使用できるようにすることです。次に、ユーザーが独自のリリースパイプラインを定義する手順の概要を示します。これらの手順の実行方法については、Codarのオンラインヘルプを参照してください。

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。



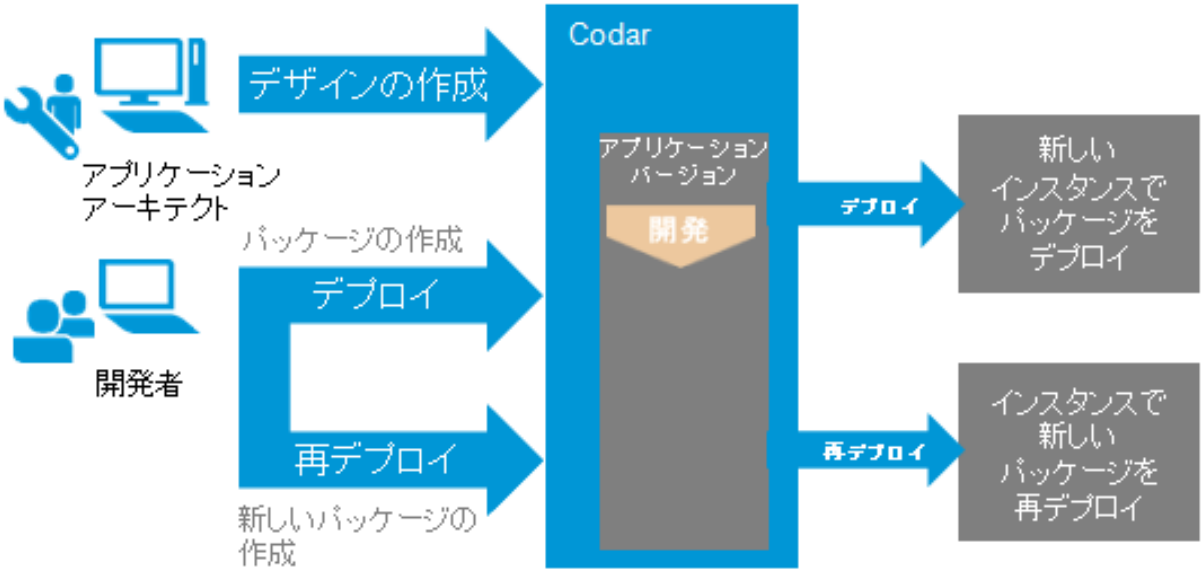
ユースケース: パッケージのデプロイと再 デプロイ

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 CLI Reference』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。

Codar Community Edition

Codarのフル機能の30日間試用版が、Dockerイメージで利用できます。

Community Edition試用版

Codar Community Editionを使用するには、製品登録とアクティブ化が必要です。

ユーザー登録が済むと、30日間の使用版ライセンスの製品のアクティブ化コードが発行されます。電子メールで送信されるリンクをたどることでオンラインで製品をアクティブ化することも、アクティブ化コードを使用して製品をオンラインでアクティブ化することもできます。

ドキュメントのフィードバックの送信

本ドキュメントについてのご意見、ご感想については、電子メールでドキュメント制作チームまでご連絡ください。このシステムで電子メールクライアントが設定されていれば、このリンクをクリックすることで、以下の情報が件名に記入された電子メールウィンドウが開きます。

フィードバック: コンセプトガイド (Codar 1.70)

本文にご意見、ご感想を記入の上、[送信] をクリックしてください。

電子メールクライアントが利用できない場合は、上記の情報をコピーしてWebメールクライアントの新規メッセージに貼り付け、clouddocs@hpe.com宛にお送りください。

お客様からのご意見、ご感想をお待ちしています。