OpenShift Container Platform 3.9

Ansible Playbook Bundle 開発ガイド

Ansible Playbook Bundle (APB) を使用した開発

Last Updated: 2018-08-14

OpenShift Container Platform 3.9 Ansible Playbook Bundle 開発ガイド

Ansible Playbook Bundle (APB) を使用した開発

法律上の通知

概要

本書では、APB の設計概念およびワークフローについての概要を説明します。

目次

第1章 はじめに **5**
1.1. 本ガイドについて 5
1.2. 設計の概要 5
1.3. ワークフロー 5
1.3.1. 準備 5
1.3.1.1. APB の初期化 5
1.3.1.2. APB 仕様ファイル 6
1.3.1.3. アクション 6
1.3.2. ビルド 7
1.3.3. デプロイ 7

第2章 CLI ツール **11**
2.1. 概要 11
2.2. ツールのインストール 11
2.2.1. 要件 11
2.2.1.1. Docker デーモン 11
2.2.1.2. アクセスパーミッション 11
2.2.2. RPM でのインストール 12
2.2.3. インストールの検証 12
2.3. 通常のワークフロー 12
2.3.1. ローカルレジストリー 12
2.3.2. リモートレジストリー 13
2.4. APB 作成コマンド 13
2.4.1. init 13
説明 13
使用法 13
引数 13
オプション 13
例 14
2.4.2. prepare 15
説明 15
使用法 15
オプション 15
例 16
2.4.3. build 16
説明 16
使用法 16
オプション 16
例 16
2.4.4. push 17
説明 17
使用法 17
オプション 17
例 17
2.4.5. test 18
説明 18
使用法 18
オプション 18
例 18
2.5. ブローカーのユーティリティーコマンド 18
2.5.1. list 18

説明 18
使用法 18
オプション 18
例 19
2.5.2. bootstrap 19
説明 19
使用法 19
オプション 19
例 20
2.5.3. remove 20
説明 20
使用法 20
オプション 20
例 21
2.5.4. relist 21
説明 21
使用法 21
オプション 21
例 22
2.6. その他のコマンド 22
2.6.1. help 22
説明 22
使用法 22
例 22

第3章 APB の作成 23
3.1. APB の作成: 作業の開始 23
3.1.1. 概要 23
3.1.2. 作業開始前の準備 23
3.1.3. APB の初回作成 23
3.1.4. アクションの追加 25
3.1.4.1. プロビジョニング 26
3.1.4.1.1. デプロイメント設定の作成 28
3.1.4.1.2. サービスの作成 30
3.1.4.1.3. ルートの作成 31
3.1.4.2. プロビジョニング解除 32
3.1.4.2.1. バインド 33
3.1.4.2.1.1. 準備 33
3.1.4.2.1.2. UI からの実行 38
3.1.4.2.2. テスト 43
3.1.4.2.2.1. テストアクションの作成 43
3.1.4.2.2.2. 検証ロールの作成 44
3.1.4.2.2.3. テスト結果の保存 44
3.1.4.2.2.4. テストアクションの実行 45
3.2. APB の作成: 参照 45
3.2.1. 概要 45
3.2.2. ディレクトリー構造 46
3.2.3. APB 仕様ファイル 46
3.2.3.1. 上部の構造 47
3.2.3.2. メタデータ 47
3.2.3.3. プラン 48
3.2.3.4. プランメタデータ 48
3.2.3.5. パラメーター 48

3.2.3.6. APB 仕様のバージョン管理 49
3.2.3.6.1. メジャーバージョン 50
3.2.3.6.2. マイナーバージョン 50
3.2.4. Dockerfile 50
3.2.5. APB アクション (Playbook) 50
3.2.6. 共通リソースの使用 51
3.2.6.1. サービス 51
3.2.6.2. デプロイメント設定 52
3.2.6.3. ルート 52
3.2.6.4. 永続ボリューム 53
3.2.7. オプション変数 54
3.2.8. 制限付き SCC の使用 54
3.2.9. ConfigMap の APB 内での使用 55

第1章 はじめに

1.1. 本ガイドについて

本書では、Ansible Playbook Bundles (APB) の設計概念およびワークフローについての概念を説明します。また、**apb** CLI ツールのインストールおよび使用方法について説明し、独自の APB を作成する際に使用できるチュートリアルおよび参考情報を提供します。

1.2. 設計の概要

APB は Nulecule および Atomicapp プロジェクトのいくつかの概念、つまり対象アプリケーションのデプイメントのオーケストレーションのみを目的とする有効期限の短いコンテナーの概念に基づく軽量アプリケーション定義です。APB の場合に、この有効期限の短いコンテナーとは APB 自体のことです。このコンテナーには、Ansible ランタイム環境と Playbook、ロールおよび追加の依存関係などのオーケストレーションを支援する必要なファイルが含まれます。

OpenShift Ansible Broker (OAB) は、APB で定義されるアプリケーションを管理する Open Service Broker (OSB) API の実装です。OAB は、OpenShift Container Platform 3.7 よりデフォルトでサポートされ、デプロイされています。

APB の仕様は軽量であることが意図されており、パラメーターなどの情報をキャプチャーしてアプリケーションに渡すためのいくつかの名前付き Playbook と 1 つのメタデータファイルで構成されます。

1.3. ワークフロー

APB ワークフローは以下のステップに分けられます。

1. 準備
    a. APB の初期化
    b. APB 仕様ファイル
    c. アクション (プロビジョニング、プロビジョニング解除、バインド、バインド解除)
2. ビルド
3. デプロイ

1.3.1. 準備

APB をビルドし、デプロイする前に APB のディレクトリー構造および仕様ファイルを準備する必要があります。「作業の開始」のトピックでは、APB の初回作成時の段階的なチュートリアルを提供しています。以下のセクションでは、このワークフローについて簡単に説明しています。

1.3.1.1. APB の初期化

`apb init` コマンドは、APB に必要なスケルトンのディレクトリー構造といくつかの必要なファイル (**apb.yml** 仕様ファイルなど) を作成します。

以下は、APB のディレクトリー構造のサンプルを示しています。

ディレクトリー構造

```
example-apb/
├── Dockerfile
├── apb.yml
└── roles/
|    └── example-apb-openshift
|        ├── defaults
|        |    └── main.yml
|        └── tasks
|            └── main.yml
└── playbooks/
    └── provision.yml
    └── deprovision.yml
    └── bind.yml
    └── unbind.yml
```

1.3.1.2. APB 仕様ファイル

APB 仕様ファイル (**apb.yml**) は特定のアプリケーション用に編集する必要があります。たとえば、**apb init** を実行後のデフォルトの仕様ファイルは以下のようになります。

```
version: 1.0
name: my-test-apb
description: This is a sample application generated by apb init
bindable: False
async: optional
metadata: 1
  displayName: my-test
plans:
  - name: default
    description: This default plan deploys my-test-apb
    free: True
    metadata: {}
    parameters: [] 2
```

1 **metadata** フィールドはオプションで、OpenShift Container Platform サービスカタログと統合する場合に使用されます。

2 いずれのパラメーターも持たない APB の場合、**parameters** フィールドは空白になります。

注記

完全に定義された APB 仕様ファイルのサンプルについては、「参考情報」のトピックを参照してください。

1.3.1.3. アクション

以下は APB のアクションです。APB は少なくともプロビジョニングおよびプロビジョニング解除のアクションを実装する必要があります。

provision.yml

アプリケーションのクラスターへのインストールを処理するために呼び出される Playbook。

deprovision.yml

アンインストールを処理するために呼び出される Playbook。

bind.yml

認証情報の生成など、別のサービスがこのサービスを使用するために使用するアクセスを付与するための Playbook。

unbind.yml

このサービスへのアクセスを取り消すための Playbook。

test.yml

(オプション) APB が有効であることをテストするための Playbook。

必要な名前付き Playbook は OSB API で定義されるメソッドに対応します。たとえば、OAB が APB をプロビジョニングする必要がある場合、これは **provision.yml** を実行します。

必要な名前付き Playbook が生成された後は、ファイルを直接使用してアプリケーションの管理をテストできます。開発者はこのファイルのディレクトリーを使用して、期待する動作が得られるまで調整と実行を繰り返す必要があるかもしれません。開発者は Ansible を Playbook および必要な変数を使って直接起動して Playbook をテストすることができます。

1.3.2. ビルド

ビルドのステップは、名前付き Playbook から配布するコンテナーイメージをビルドします。パッケージ化により、Ansible ランタイムを含むベースイメージと Playbook の実行に必要な Ansible アーティファクトおよびその他の依存関係が組み合わされます。

結果として、いくつかの引数を取るための **ENTRYPOINT** が設定されたコンテナーイメージがビルドされます。それらの 1 つにはプロビジョニングおよびプロビジョニング解除などの実行メソッドが含まれます。

図1.1 APB のビルド

1.3.3. デプロイ

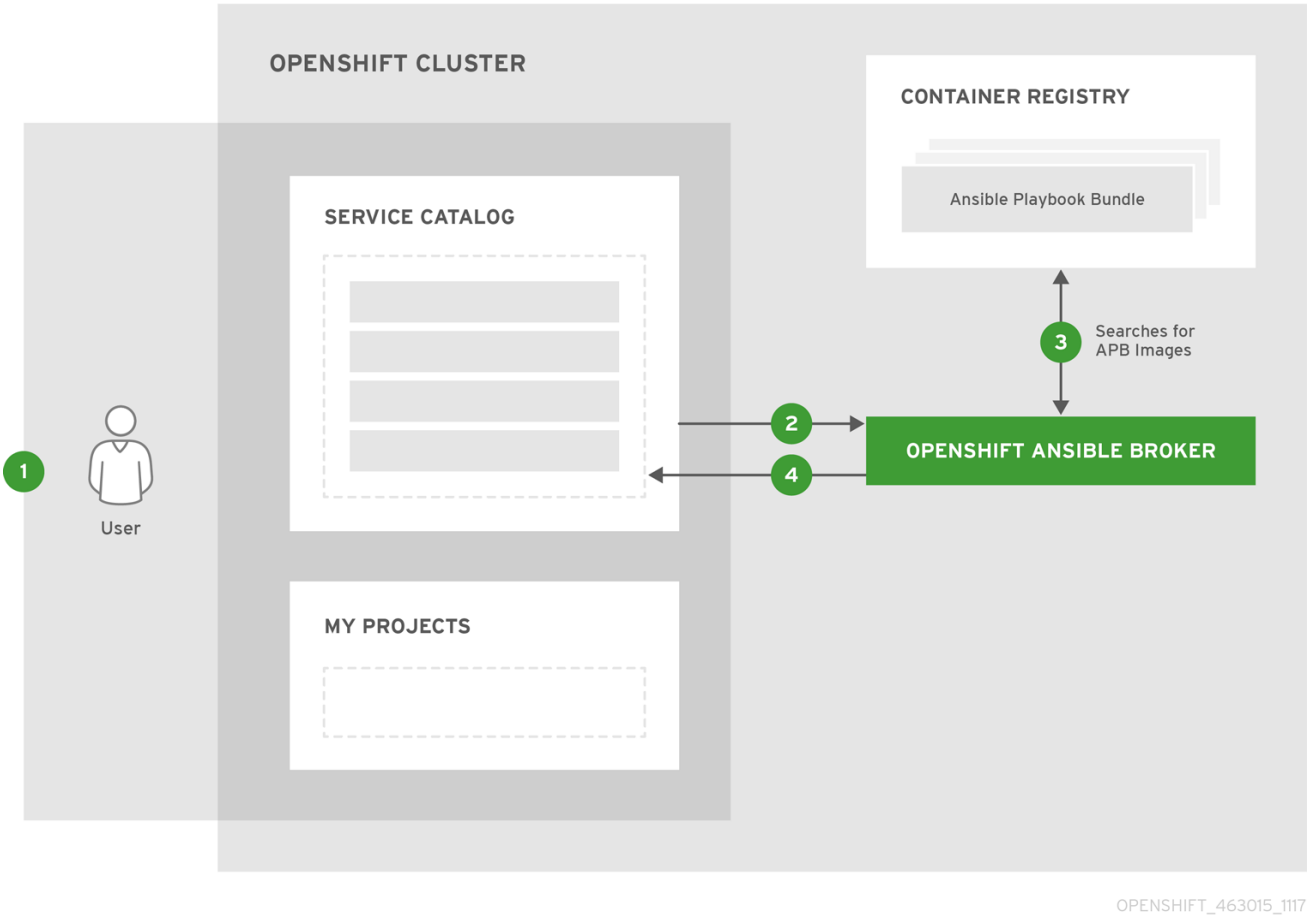
APB のデプロイにより、コンテナーが起動し、必要な変数と共に実行される Playbook の名前が渡されます。OAB を経由しなくても APB を直接起動できます。それぞれの APB はパッケージ化されるため、その **ENTRYPOINT** が実行時に Ansible を起動します。このコンテナーの有効期限は短く設定され、アプリケーションを管理し、終了するための Ansible Playbook を実行します。

通常の APB デプロイでは、APB コンテナーは、Ansible ロールを実行する **provision.yml** Playbook を実行してアプリケーションをプロビジョニングします。このロールは、**`oc create`** コマンドを呼び出すか、または Ansible モジュールを利用して OpenShift Container Platform リソースを作成します。最

終的な結果として、APB は Ansible を実行して OpenShift Container Platform と通信し、対象アプリケーションのプロビジョニングのオーケストレーションを実行します。

以下の図は、ユーザーが利用可能な APB の一覧を検出し、次に選択した APB をプロジェクトにプロビジョニングする 2 つのフェーズで構成されるデプロイメントのフローを示しています。

図1.2 利用可能な APB の一覧表示



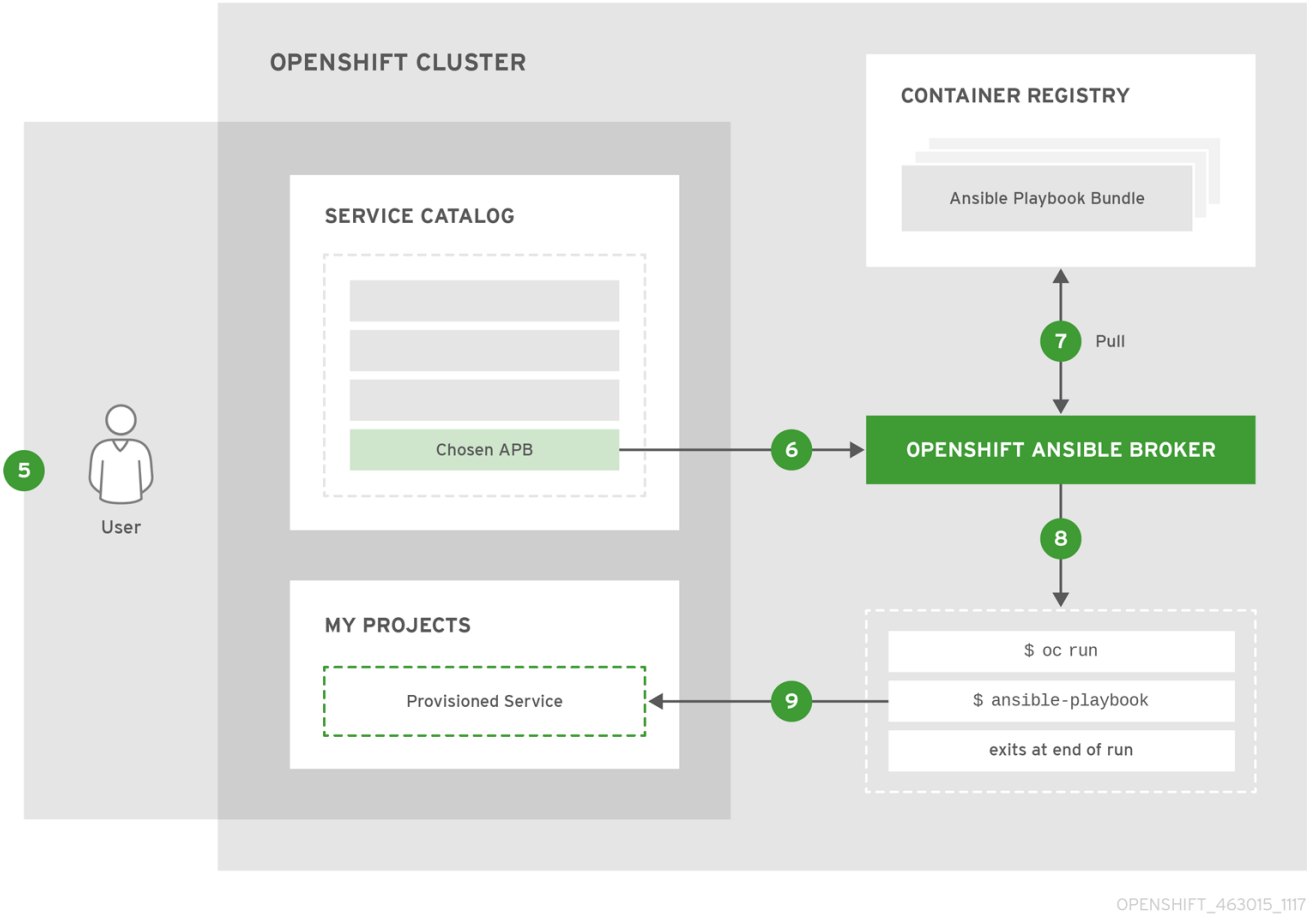
1 OpenShift Container Platform ユーザーはサービスをプロジェクトにプロビジョニングすることを検討しているので、OpenShift Container Platform UI (Web コンソールまたは CLI) にアクセスしてサービスカタログと対話し、すでに利用可能な APB を検出します。

2 サービスカタログは、ユーザーに表示するために OAB から APB の一覧を要求します。

3 OAB は設定済みのすべてのコンテナーレジストリー (クラスターの OpenShift Container レジストリーまたはその他のリモートレジストリー) で APB (**`LABEL=apb-1.0`** などの特定ラベルの付いたイメージ) を検索します。

4 OAB は検出された一覧をサービスカタログに返し、ユーザーはこの一覧を OpenShift Container Platform UI で表示できます。

図1.3 選択した APB のデプロイ



5 ユーザーはサービスカタログで提供される検出された一覧から APB を選択できるようになります。

6 サービスカタログは、ユーザーが選択した APB の使用について要求した OAB と通信します。

7 OAB は適切なコンテナーレジストリーからイメージプルを開始します。

8 イメージのプル後に、OAB はアプリケーションのオーケストレーションのロジックを APB に送ります。サービスは APB コンテナーをいくつかのパラメーターで実行してデプロイされます。これを実行するには、以下のコマンドを一時的な namespace の OpenShift Container Platform クラスターに対して実行します。

```
$ oc run $IMAGE $METHOD $VARS ansible-playbook ${METHOD}.yaml ${VARS}
```

このコマンドは、以下のように分けられます。

1. **oc run** コマンドは APB イメージを実行します。
2. 結果として作成される有効期限の短いコンテナーでは、Ansible は、必要なアクションを実行するための Playbook (**provision.yaml** など) を実行する **ansible-playbook** コマンドを使用して起動します。これにより、OpenShift Container Platform リソースがユーザーのプロジェクトに作成されます。
3. コンテナーは実行が終了すると終了し、一時的な namespace は削除されます。

9 結果として、ユーザーは **OpenShift Container Platform UI** から要求したサービスがプロジェクトに正常にプロビジョニングされていることを確認できます。

第2章 CLI ツール

2.1. 概要

apb CLI ツールを使用する Ansible Playbook Bundle (APB) の作成者は、APB を作成し、ビルドしてこれをコンテナーレジストリーに公開できます。これにより、ベストプラクティスが実施され、詳細事項が処理されることでデプロイが容易になります。

2.2. ツールのインストール

2.2.1. 要件

2.2.1.1. Docker デーモン

docker デーモンは正常にインストールされており、システム上で実行されている必要があります。

2.2.1.2. アクセスパーミッション

apb ツールは、ユーザーにトークンを使用するクラスターユーザーとしてログインすることを要求します。デフォルトの **system:admin** システムユーザーはツールの認証に使用できるトークンを持たないために適切ではありません。さらに、**apb** ツールの機能全体を許可するには、数多くのローカルロール (プロジェクト単位) およびクラスターロール (クラスター全体) が存在している必要があります (「Cluster and Local RBAC 」を参照してください)。

最も簡単なオプションとして、ユーザーに **cluster-admin** クラスターロールを持たせることができます。このロールを別のユーザーに追加するには、このパーミッションをすでに持つユーザー (**system:admin** デフォルトシステムユーザーなど) として以下を実行することができます。

警告

これはクラスターの **root** であり、開発設定でのみ使用する必要があります。

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user>
$ oc login -u <user> <openshift_server>
```

さらに厳密にパーミッションが設定された環境が必要な場合、デフォルトで提供される OpenShift テンプレートにより **developer** というユーザーのロールが付与されます。このテンプレートは、各種ロールを作成できるパーミッションのあるユーザーによって実行される必要があります。**developer** ユーザーにはこのようなパーミッションはありませんが、**system:admin** ユーザーにはこれを実行するためのパーミッションがあります。

テンプレートを実行するには、以下を実行します。

1. **openshift-permissions.template.yaml** ファイルをローカルにダウンロードします。
2. 以下のコマンドを実行します。

```
$ oc process -f openshift-permissions.template.yaml \
   -p BROKER_NAMESPACE=openshift-ansible-service-broker \
   [-p USER=<your_desired_user>] \(1)
   | oc create -f -
```

(1) デフォルトで、テンプレートは **developer** ユーザーのロールを付与します。オプションで **-p** フラグを使用して、デフォルト値を必要なユーザーで上書きすることができます。

2.2.2. RPM でのインストール

APB CLI ツールは **apb** パッケージによって提供されます。これは **rhel-7-server-ose-3.7-rpms** チャンネルで利用できます。

```
$ sudo yum install apb
```

2.2.3. インストールの検証

apb help を実行して、ツールが適切にインストールされていることを確認します。

```
$ apb help
usage: apb [-h] [--debug] [--project BASE_PATH]
           {init,help,prepare,push,bootstrap,list,remove,build} ...

APB tooling for assisting in building and packaging APBs.

optional arguments:
  -h, --help            show this help message and exit
  --debug               Enable debug output
  --project BASE_PATH, -p BASE_PATH
                        Specify a path to your project. Defaults to CWD.

subcommand:
  {init,help,prepare,push,bootstrap,list,remove,build}
    init                Initialize the directory for APB development
    help                Display this help message
    prepare             Prepare an ansible-container project for APB
packaging
    push                Push local APB spec to an OAB
    bootstrap           Tell OAB to reload APBs from the
                        container repository
    list                List APBs from the target OAB
    remove              Remove APBs from the target OAB
    build               Build and package APB container
```

2.3. 通常のワークフロー

2.3.1. ローカルレジストリー

OpenShift Container レジストリーを使用して APB を調達するには、OpenShift Ansible Broker を **local_openshift** タイプのレジストリーアダプターを使用するように設定する必要があります。詳細は、「config」のセクションを参照してください。

```
$ apb init my-new-apb
$ cd my-new-apb
$ apb build
$ apb push
$ apb list
```

APB をホストするためにデフォルトの **openshift** namespace 以外の namespace を使用する場合は、以下のコマンドを使用できます。

```
$ apb push --namespace <namespace>
```

2.3.2. リモートレジストリー

OAB は、**docker.io/ansibleplaybookbundle** または独自の個人アカウントなどの、リモートレジストリーや組織を使用するように設定することもできます。これを APB を開発するために使用するには、APB をビルドし、リモートレジストリーにプッシュしてから **bootstrap** を実行して APB を再ロードできます。

```
$ apb init my-new-apb
$ cd my-new-apb
$ apb build --tag docker.io/my-org/my-new-apb
$ docker push docker.io/my-org/my-new-apb
$ apb bootstrap
$ apb list
```

2.4. APB 作成コマンド

2.4.1. `init`

説明
新規 APB のディレクトリー構造を初期化します。さらに、適切なデフォルト値が設定された新規 APB のサンプルファイルを作成します。

使用法

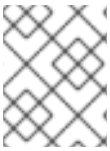
```
$ apb init [OPTIONS] NAME
```

引数
NAME: 作成される APB およびディレクトリーの名前です。

オプション

オプション (短縮形表記)	説明
--help, -h	ヘルプメッセージを表示します。
--force	ディレクトリーの再初期化および上書きを強制的に実行します。
--async {required,optional,unsupported}	アプリケーションの非同期操作を指定します。通常は **optional** にデフォルト設定されます。

オプション (短縮形表記)	説明
--bindable	アプリケーションをバインド可能な設定で生成します。
--skip-provision	プロビジョニング Playbook およびロールを生成しません。
--skip-deprovision	プロビジョニング解除 Playbook およびロールを生成しません。
--skip-bind	バインド Playbook およびロールを生成しません。
--skip-unbind	バインド解除 Playbook およびロールを生成しません。
--skip-roles	ロールを生成しません。



注記

非同期バインドおよびバインド解除は実験的な機能であり、デフォルトではサポートされておらず、有効にされていません。

例
ディレクトリー **my-new-apb** を作成します。

```
$ apb init my-new-apb
# my-new-apb/
# ├── apb.yml
# ├── Dockerfile
# ├── playbooks
# │   ├── deprovision.yml
# │   └── provision.yml
# └── roles
#     ├── deprovision-my-new-apb
#     │   └── tasks
#     │       └── main.yml
#     └── provision-my-new-apb
#         └── tasks
#             └── main.yml
```

ディレクトリー **my-new-apb** を作成しますが、プロビジョニング解除 Playbook およびロールの生成を省略します。

```
$ apb init my-new-apb --skip-deprovision
# my-new-apb/
# ├── apb.yml
# ├── Dockerfile
# ├── playbooks
# │   └── provision.yml
```

```
# └── roles
#     └── provision-my-new-apb
#         └── tasks
#             └── main.yml
```

ディレクトリー **my-new-apb** を作成し、古いバージョンを上書きします。APB はバインド可能になるように設定され、**async** をオプションに設定します。

```
$ apb init my-new-apb --force --bindable --async optional
# my-new-apb/
# ├── apb.yml
# ├── Dockerfile
# ├── playbooks
# │   ├── bind.yml
# │   ├── deprovision.yml
# │   ├── provision.yml
# │   └── unbind.yml
# └── roles
#     ├── bind-my-new-apb
#     │   └── tasks
#     │       └── main.yml
#     ├── deprovision-my-new-apb
#     │   └── tasks
#     │       └── main.yml
#     ├── provision-my-new-apb
#     │   └── tasks
#     │       └── main.yml
#     └── unbind-my-new-apb
#         └── tasks
#             └── main.yml
```

2.4.2. prepare

説明
APB を **base64** エンコーディングでコンパイルし、これを **Dockerfile** のラベルとして作成します。

これにより、OAB はイメージをダウンロードせずにレジストリーから APB メタデータを読み取ることができます。このコマンドは APB ディレクトリー内から実行される必要があります。**build** コマンドを実行すると、**prepare** も自動的に実行されます。つまり、通常は **prepare** 自体を実行する必要はありません。

使用法

```
$ apb prepare [OPTIONS]
```

オプション

オプション (短縮形表記)	説明
--help, -h	ヘルプメッセージを表示します。
--dockerfile DOCKERFILE, -f DOCKERFILE	APB 仕様を、**Dockerfile** という名前のファイルの代わりにターゲットファイル名に書き込みます。

例
Dockerfile に仕様フィールドのラベルを作成します。

```
$ apb prepare
```

Dockerfile-custom に仕様フィールドのラベルを作成します。

```
$ apb prepare --dockerfile Dockerfile-custom
```

2.4.3. build

説明
APB のイメージをビルドします。

タグを使って **apb prepare** および **docker build** を実行するのと同様です。

使用法

```
$ apb build [OPTIONS]
```

オプション

オプション (短縮形表記)	説明
--help, -h	ヘルプメッセージを表示します。
--tag TAG	ビルドイメージのタグを **<registry>/<org>/<name>** 形式で文字列に設定します。
--registry	イメージのタグのレジストリー部分です (例: **docker.io**)。
--org, -o	イメージのタグのユーザーまたは組織の部分です。

例
イメージをビルドし、**apb.yml** の名前フィールドをタグとして使用します。

```
$ apb build
```

イメージをビルドし、タグ **docker.io/my-org/my-new-apb** を使用します。

```
$ apb build --tag docker.io/my-org/my-new-apb
```

イメージをビルドし、タグ **docker.io/my-org/<my-apb-name>** を使用します。

```
$ apb build --registry docker.io --org my-org
```

ファイル **Dockerfile-custom** を **Dockerfile** 定義として使用してイメージをビルドします。

```
$ apb build --dockerfile Dockerfile-custom
```

2.4.4. push

説明
APB を OpenShift Container レジストリーに、またはこれが OAB による読み取りが行われるブローカーのモックレジストリーにアップロードします。

ブローカーのモックレジストリーを使用する場合、仕様はアップロードされ、OpenShift Container Platform に表示されますが、通常 OpenShift Container Platform はイメージをレジストリーからプルします。このレジストリーは通常は **oc cluster up** が実行されるレジストリーになります。

OpenShift Container レジストリーを使用する場合、イメージは OpenShift Container Platform に直接アップロードされます。

使用法

```
$ apb push [OPTIONS]
```

オプション

オプション (短縮形表記)	説明
--help, -h	ヘルプメッセージを表示します。
--broker BROKER_URL	OAB へのルートです。
--namespace NAMESPACE	OpenShift Container レジストリーにプッシュする namespace です。
--openshift, -o	OpenShift Container レジストリーを使用します。
--dockerfile DOCKERFILE, -f DOCKERFILE	内部レジストリーイメージをビルドするための **Dockerfile** です。通常は **Dockerfile** に設定されますが、任意のファイル名に設定することができます。
--secure	OAB へのセキュアな接続を使用します。
--username USERNAME	ブローカーの通信で使用される基本認証ユーザー名です。
--password PASSWORD	ブローカーの通信で使用される基本認証パスワードです。
--no-relist	APB のブローカーへのプッシュ後にカタログを再度一覧表示しません。
--broker-name	ServiceBroker Kubernetes リソースの名前です。

例
OAB 開発エンドポイントにプッシュします。

```
$ apb push
```

ローカルの OpenShift Container レジストリーにプッシュします。

```
$ apb push
```

namespace **myproject** の下のローカルの OpenShift Container レジストリーにプッシュします。

```
$ apb push --namespace myproject
```

2.4.5. test

説明
APB 単体テストを実行します。

使用法

```
$ apb test [OPTIONS]
```

オプション

オプション (短縮形表記)	説明
--help, -h	ヘルプメッセージを表示します。
--tag TAG	ビルドイメージのタグを **<registry>/<org>/<name>** 形式で文字列に設定します。

例
テストを実行します。

```
$ apb test
```

テストを実行しますが、ビルドされたイメージの特定のタグを使用します。

```
$ apb test --tag docker.io/my-org/my-new-apb
```

2.5. ブローカーのユーティリティーコマンド

2.5.1. list

説明
ブローカーが読み込んだすべての APB を一覧表示します。

使用法

```
$ apb list [OPTIONS]
```

オプション

オプション (短縮形表記)	説明
--help, -h	ヘルプメッセージを表示します。
--broker BROKER_URL	OAB へのルートです。
--secure	OAB へのセキュアな接続を使用します。
--verbose, -v	OAB から詳細な仕様情報を出力します。
--output {yaml,json}, -o {yaml,json}	詳細な出力形式を yaml (デフォルト) または json で指定します。
--username BASIC_AUTH_USERNAME, -u BASIC_AUTH_USERNAME	使用される基本認証ユーザー名を指定します。
--password BASIC_AUTH_PASSWORD, -p BASIC_AUTH_PASSWORD	使用される基本認証パスワードを指定します。

例
名前、ID および説明を含む APB の基本的な一覧です。

```
$ apb list
```

詳細な、読み取りが容易な仕様を一覧表示します。

```
$ apb list -v
```

すべての JSON 出力を一覧表示します。

```
$ apb list -v -o json
```

2.5.2. bootstrap

説明
OAB がすべての APB をレジストリーからリロードすることを要求します。

使用法

```
$ apb bootstrap [OPTIONS]
```

オプション

オプション (短縮形表記)	説明
--help, -h	ヘルプメッセージを表示します。
--broker BROKER_URL	OAB へのルートです。

オプション (短縮形表記)	説明
--secure	OAB へのセキュアな接続を使用します。
--no-relist	ブローカーのブートストラップにカタログを再度一覧表示しません。
--username BASIC_AUTH_USERNAME, -u BASIC_AUTH_USERNAME	使用される基本認証ユーザー名を指定します。
--password BASIC_AUTH_PASSWORD, -p BASIC_AUTH_PASSWORD	使用される基本認証パスワードを指定します。
--broker-name BROKER_NAME	ServiceBroker Kubernetes リソースの名前です。

例
APB の基本的なリロード:

```
$ apb bootstrap
```

2.5.3. remove

説明
1 つ (またはすべての) APB を OAB から削除します。

使用法

```
$ apb remove [OPTIONS]
```

オプション

オプション (短縮形表記)	説明
--help, -h	ヘルプメッセージを表示します。
--broker BROKER_URL	OAB へのルートです。
--secure	OAB へのセキュアな接続を使用します。
--all	すべての保存された APB を削除します。
--id ID	削除する APB の ID です。
--secure	OAB へのセキュアな接続を使用します。
--username BASIC_AUTH_USERNAME, -u BASIC_AUTH_USERNAME	使用される基本認証ユーザー名を指定します。

オプション (短縮形表記)	説明
--password BASIC_AUTH_PASSWORD, -p BASIC_AUTH_PASSWORD	使用される基本認証パスワードを指定します。
--no-relist	削除後にカタログを再度一覧表示しません。

例
ID を使用して APB を削除します。

```
$ apb remove --id ca91b61da8476984f18fc13883ae2fdb
```

注記

APB の ID が必要な場合は、以下を使用します。

```
$ apb list
ID                                NAME
DESCRIPTION
ca91b61da8476984f18fc13883ae2fdb  dh-etherpad-apb          Note
taking web application
```

すべての APB を削除します。

```
$ apb remove --all
```

2.5.4. relist

説明
サービスカタログに対し、ブローカーに一致する提供されたサービスを再度一覧表示するよう強制します。

使用法

```
$ apb relist [OPTIONS]
```

オプション

オプション (短縮形表記)	説明
--help, -h	ヘルプメッセージを表示します。
--broker-name BROKER_NAME	ServiceBroker Kubernetes リソースの名前です。
--secure	OAB へのセキュアな接続を使用します。
--username BASIC_AUTH_USERNAME, -u BASIC_AUTH_USERNAME	使用される基本認証ユーザー名を指定します。

オプション (短縮形表記)	説明
--password BASIC_AUTH_PASSWORD, -p BASIC_AUTH_PASSWORD	使用される基本認証パスワードを指定します。

例

```
$ apb relist
```

2.6. その他のコマンド

2.6.1. help

説明
ヘルプメッセージを表示します。

使用法

```
$ apb help
```

例

```
$ apb help
```

```
$ apb -h
```

第3章 APB の作成

3.1. APB の作成: 作業の開始

3.1.1. 概要

このチュートリアルでは、いくつかのサンプル Ansible Playbook Bundle (APB) の作成を行います。APB でプロビジョニング、プロビジョニング解除、バインドおよびバインド解除を可能にするアクションを作成します。APB の設計についての詳細は、「設計」のトピックを参照してください。APB の作成についてのさらに詳細な情報は、「参照」のトピックを参照してください。

注記

このチュートリアルの残りの部分では、括弧のマークされた項目を独自の情報で置き換えます。たとえば、**<host>:<port>** を **172.17.0.1.nip.io:8443** などに置き換えます。

3.1.2. 作業開始前の準備

独自の APB を作成する前に、開発環境をセットアップする必要があります。

1. OpenShift Container Platform クラスターにアクセスできることを確認します。クラスターはサービスカタログと、OpenShift Container Platform 3.7 以降サポートされている OpenShift Ansible broker (OAB) の両方を実行している必要があります。

2. APB ツールを「CLI ツール」のトピックで説明されているようにインストールします。検証するには、**apb help** コマンドを実行して、有効な応答の有無を確認します。

3.1.3. APB の初回作成

このチュートリアルでは、コンテナー化された hello world アプリケーションの APB を作成します。ここでは、APB **hello-world-apb** をミラーリングする基本的な APB を扱います。

1. 最初のタスクとして、**apb** CLI ツールを使用して APB を初期化します。これにより、APB のスケルトンが作成されます。ここで使用するコマンドは単純なコマンドです。

```
$ apb init my-test-apb
```

初期化の後に、以下の構造を確認できます。

```
my-test-apb/
├── apb.yml
├── Dockerfile
├── playbooks
│   ├── deprovision.yml
│   └── provision.yml
└── roles
    ├── deprovision-my-test-apb
    │   └── tasks
    │       └── main.yml
    └── provision-my-test-apb
        └── tasks
            └── main.yml
```

apb.yml (APB 仕様ファイル) および **Dockerfile** の 2 つのファイルがルートディレクトリーに作成されています。これらは APB に必要な最低限のファイルです。APB 仕様ファイルについての詳細は、「参照」のトピックを参照してください。ここでは、**Dockerfile** で実行できることについても説明されています。

apb.yml

```
version: 1.0
name: my-test-apb
description: This is a sample application generated by apb init
bindable: False
async: optional
metadata:
  displayName: my-test
plans:
  - name: default
    description: This default plan deploys my-test-apb
    free: True
    metadata: {}
    parameters: []
```

Dockerfile

```
FROM ansibleplaybookbundle/apb-base

LABEL "com.redhat.apb.spec"=\

COPY playbooks /opt/apb/actions
COPY roles /opt/ansible/roles
RUN chmod -R g=u /opt/{ansible,apb}
USER apb
```

2. **Dockerfile** では、2 つの更新を実行します。

   a. **FROM** 命令を変更して、イメージを Red Hat Container Catalog から使用できるようにします。最初の行が表示されるはずです。

   ```
   FROM openshift3/apb-base
   ```

   b. **LABEL** 命令の **com.redhat.apb.spec** を base64 でエンコードされたバージョンの **apb.yml** で更新します。これを実行するには、**apb prepare** を実行します。

   ```
   $ cd my-test-apb
   $ apb prepare
   ```

   これにより、以下のように **Dockerfile** が更新されます。

   Dockerfile

   ```
   FROM openshift3/apb-base

   LABEL "com.redhat.apb.spec"=\
   "dmVyc2lvbjogMS4wCm5hbWU6IG15LXRlc3QtYXBiCmRlc2NyaXB0aW9uOiBUaGlz
   IGlzIGEgc2Ft\
   ```

```
cGxlIGFwcGxpY2F0aW9uIGdlbmVyYXRlZCBieSBhcGIgaW5pdApiaW5kYWJsZTogR
mFsc2UKYXN5\
bmM6IG9wdGlvbmFsCm1ldGFkYXRhOgogIGRpc3BsYXlOYW1lOiBteS10ZXN0CnBsY
W5zOgogIC0g\
bmFtZTogZGVmYXVsdAogICAgZGVzY3JpcHRpb246IFRoaXMgZGVmYXVsdCBwbGFuI
GRlcGxveXMg\
bXktdGVzdC1hcGIKICAgIGZyZWU6IFRydWUKICAgIG1ldGFkYXRhOiB7fQogICAgc
GFyYW1ldGVy\
czogW10="

COPY playbooks /opt/apb/actions
COPY roles /opt/ansible/roles
RUN chmod -R g=u /opt/{ansible,apb}
USER apb
```

3. この時点で、ビルド可能な完全に形成された APB が使用できるようになります。**apb prepare** の使用を省略した場合でも、**apb build** コマンドがイメージのビルド前に APB を作成します。

```
$ apb build
```

4. 新規の APB イメージをローカルの OpenShift Container レジストリーにプッシュできるようになります。

```
$ apb push
```

5. OAB のクエリーによって、新規の APB が一覧表示されているのを確認できます。

```
$ apb list
ID                                   NAME            DESCRIPTION
< ------------ ID ------------->  dh-my-test-apb  This is a sample
application generated by apb init
```

同様に、OpenShift Container Platform Web コンソールにアクセスすると、サービスカタログの **All** および **Other** タブの下に **my-test-apb** という名前の新規 APB が表示されます。

3.1.4. アクションの追加

直前のセクションで作成された完全に新規の APB は、そのままの状態では多くのことを実行できないため、いくつかのアクションを追加する必要があります。サポートされているアクションには以下が含まれます。

- provision (プロビジョニング)
- deprovision (プロビジョニング解除)
- bind (バインド)
- unbind (バインド解除)
- test (テスト)

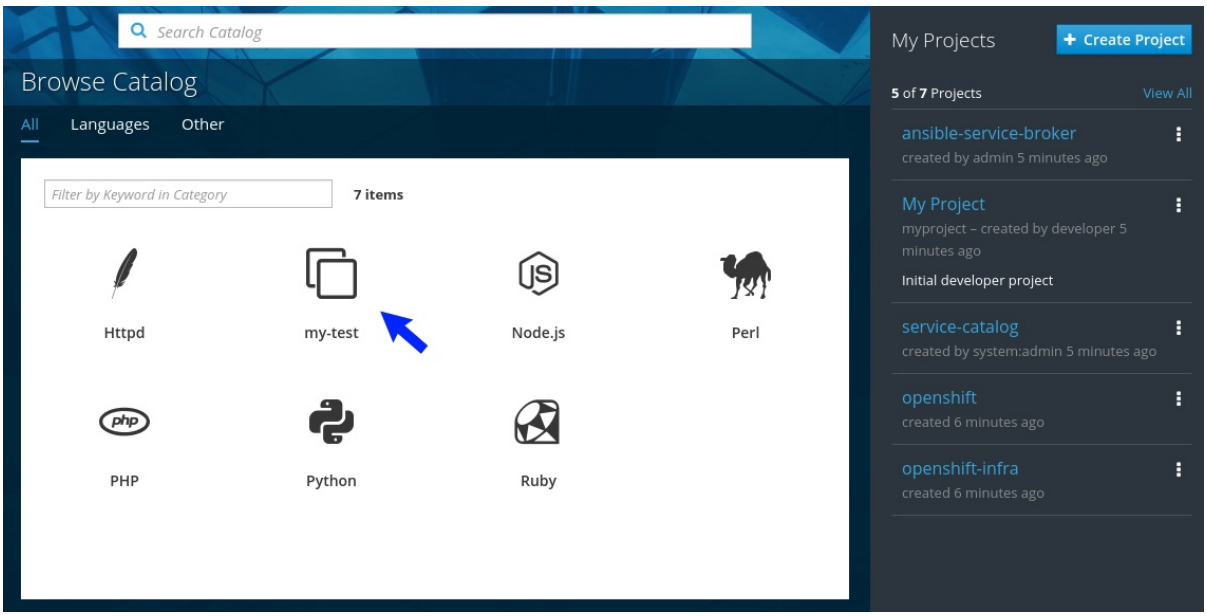
以下のセクションで上記のアクションのそれぞれを追加していきます。開始前に、以下を確認してください。

1. **oc** CLI 経由で OpenShift Container Platform クラスターにログインしている。この場合は、**apb** ツールが OpenShift Container Platform および OAB と対話できます。

```
# oc login <cluster_host>:<port> -u <user_name> -p <password>
```

2. OpenShift Container Platform Web コンソールにログインし、APB がカタログに一覧表示されていることを確認します。

図3.1 OpenShift Container Platform Web コンソール



3. **getting-started** という名前のプロジェクトを作成します。このプロジェクトで OpenShift Container Platform リソースをデプロイします。Web コンソールまたは CLI を使用してこれを作成できます。

```
$ oc new-project getting-started
```

3.1.4.1. プロビジョニング

apb init プロセスで、Playbook の **playbooks/provision.yml** と **roles/provision-my-test-apb** の関連ロールの、プロビジョニングタスクの 2 つの部分が表示されます。

```
my-test-apb
├── apb.yml
├── Dockerfile
├── playbooks
│   └── provision.yml (1)
└── roles
    └── provision-my-test-apb
        └── tasks
            └── main.yml (2)
```

(1) この Playbook を検査します。

(2) このロールを編集します。

playbooks/provision.yml ファイルは、プロビジョニングアクションが OAB から呼び出される際に実行される Ansible Playbook です。Playbook は変更可能ですが、ここではコードをそのままにしておきます。

playbooks/provision.yml

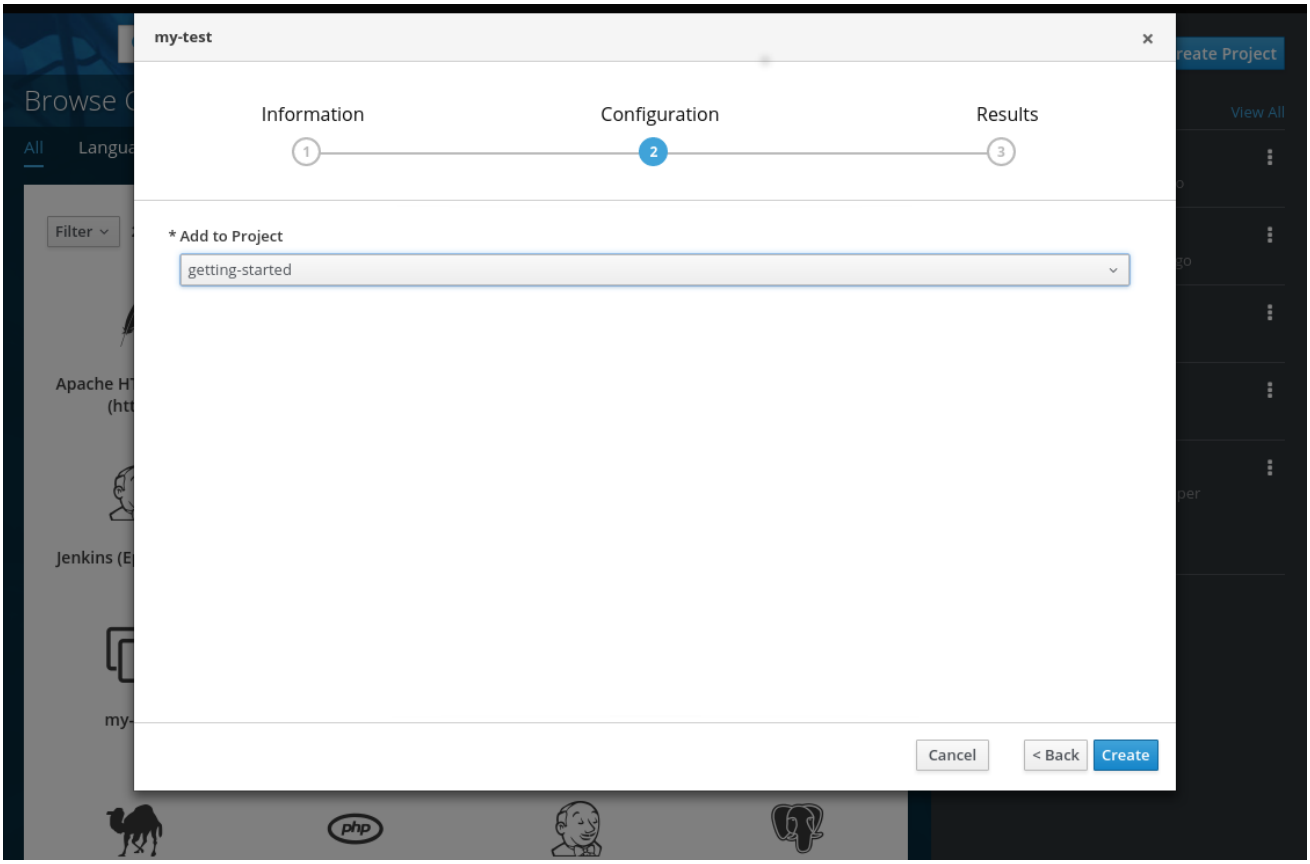
```
- name: my-test-apb playbook to provision the application
  hosts: localhost
  gather_facts: false
  connection: local
  roles:
  - role: ansible.kubernetes-modules
    install_python_requirements: no
  - role: ansibleplaybookbundle.asb-modules
  - role: provision-my-test-apb
    playbook_debug: false
```

Playbook は **localhost** で実行され、ロール **provision-my-test-apb** を実行します。この Playbook はサービスブローカーで作成されるローカルコンテナーで機能します。**ansible.kubernetes-modules** ロールによって、**kubernetes-modules** を使用して OpenShift Container Platform リソースを作成できます。**asb-modules** は OAB で使用できる追加の機能を提供します。

現時点で、このロールに設定されたタスクはありません。**roles/provision-my-test-apb/tasks/main.yml** のコンテンツには、共通のリソース作成タスクを示すコメントのみが含まれます。現時点でプロビジョニングタスクを実行できますが、実行するタスクがないために APB コンテナーが単純に起動されるのみで、何もデプロイせずに終了されることになります。

これは Web コンソールを使用し、**my-test** APB をクリックして、これを **getting-started** プロジェクトにデプロイして試してみることができます。

図3.2 my-test のプロビジョニング



プロビジョニングの実行中、新規の namespace が **dh-my-test-apb-prov-<random>** という名前で作成されます。開発モードでは、これは永続しますが、通常この namespace は正常な完了後に削除されます。APB がプロビジョニングに失敗すると、デフォルトでは namespace は永続します。

Pod リソースを参照すると、ログで APB の実行を確認することができます。Pod のログを表示するには、以下を実行します。

1. Web コンソールを使用してすべての namespace を表示し、作成日で並べ替えるか、または以下のコマンドを使用するかのいずれかを実行して namespace を検索します。

```
$ oc get ns
NAME                                    STATUS    AGE
ansible-service-broker                  Active    1h
default                                 Active    1h
dh-my-test-apb-prov-<random>            Active    4m
```

2. プロジェクトに切り替えます。

```
$ oc project dh-my-test-apb-prov-<random>
Now using project "dh-my-test-apb-prov-<random>" on server "
<cluster_host>:<port>".
```

3. Pod 名を取得します。

```
$ oc get pods
NAME              READY     STATUS      RESTARTS   AGE
<apb_pod_name>    0/1       Completed   0          3m
```

4. ログを表示します。

```
$ oc logs -f <apb_pod_name>
...
+ ansible-playbook /opt/apb/actions/provision.yml --extra-vars
'{"_apb_plan_id":"default","namespace":"getting-started"}'
PLAY [my-test-apb playbook to provision the application]
**********************
TASK [ansible.kubernetes-modules : Install latest openshift client]
*************
skipping: [localhost]
TASK [ansibleplaybookbundle.asb-modules : debug]
*****************************
skipping: [localhost]
PLAY RECAP
*********************************************************************
*
localhost                  : ok=0    changed=0    unreachable=0
failed=0
```

3.1.4.1.1. デプロイメント設定の作成

APB は少なくともアプリケーション Pod をデプロイできる必要があります。これは デプロイメント設定を指定して実行できます。

1. **provision-my-test-apb/tasks/main.yml** ファイルでコメントアウトされている最初のタスクの 1 つに、デプロイメント設定の作成があります。このコメントを解除することも、または以下を貼り付けることもできます。

注記

通常、**image:** 値を独自のアプリケーションイメージに置き換えることができます。

```
- name: create deployment config
  openshift_v1_deployment_config:
    name: my-test
    namespace: '{{ namespace }}' (1)
    labels: (2)
      app: my-test
      service: my-test
    replicas: 1 (3)
    selector: (4)
      app: my-test
      service: my-test
    spec_template_metadata_labels:
      app: my-test
      service: my-test
    containers: (5)
    - env:
      image: docker.io/ansibleplaybookbundle/hello-world:latest
      name: my-test
      ports:
      - container_port: 8080
        protocol: TCP
```

(1) デプロイメント設定が置かれる必要のある namespace を指定します。

(2) オブジェクトの編成、分類および選択に使用されます。

(3) 1 つの Pod のみを必要とすることを指定します。

(4) **selector** セクションは Pod に対する ラベルのクエリーです。

(5) この **containers** セクションは、コンテナーを TCP のポート 8080 で実行される **hello-world** アプリケーションで指定します。イメージは docker.io/ansibleplaybookbundle/hello-world に保存されます。

詳細については、『APB の作成: 参照』を参照してください。また、すべてのフィールドの詳細については、**ansible-kubernetes-modules** ドキュメントを参照してください。

2. APB をビルドし、プッシュします。

```
$ apb build
$ apb push
```

3. Web コンソールを使用して APB をプロビジョニングします。

4. プロビジョニングの後に、新規に実行される Pod および新規のデプロイメント設定が利用可能になります。OpenShift Container Platform リソースをチェックして検証します。

```
$ oc project getting-started
$ oc get all
NAME         REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/my-test   1          1         1         config

NAME           DESIRED   CURRENT   READY     AGE
rc/my-test-1   1         1         1         35s

NAME                 READY     STATUS    RESTARTS   AGE
po/my-test-1-2pw4t   1/1       Running   0          33s
```

また、Web コンソールのプロジェクトの **Overview** ページでデプロイされたアプリケーションを表示できます。

この Pod を現行の状態で使用する唯一の方法として、以下を実行します。

```
$ oc describe pods/<pod_name>
```

これにより、その IP アドレスを検索し、それに直接アクセスします。複数の Pod がある場合、それらは別個にアクセスされます。それらを単一ホストのように処理するには、次のセクションで説明されているように **サービス** を作成する必要があります。

ヒント

次に進む前にクリーンアップし、再度プロビジョニングを実行するために、**getting-started** プロジェクトを削除してから再作成するか、または新規のプロジェクトを作成することができます。

3.1.4.1.2. サービスの作成

複数の Pod を使用して負荷分散を行い、ユーザーがそれらに単一ホストとしてアクセスできるようにサービスを作成する必要になる場合があります。以下を実行します。

1. **provision-my-test-apb/tasks/main.yml** ファイルを変更し、以下を追加します。

```
- name: create my-test service
  k8s_v1_service:
    name: my-test
    namespace: '{{ namespace }}'
    labels:
      app: my-test
      service: my-test
    selector:
      app: my-test
      service: my-test
    ports:
      - name: web
        port: 80
        target_port: 8080
```

selector セクションでは、**my-test** サービスに適切な Pod を組み込むことができます。**ports** はターゲットポートを Pod から取得し (8080)、それらをサービスの単一ポート (80) として公開します。アプリケーションは 8080 で実行されていますが、これがデフォルト

の HTTP ポート 80 で利用可能になることに注意してください。

ポートの **name** フィールドでは、このポートを今後に他のリソースで使用できるように指定できます。詳細については、**k8s_v1_service** モジュールを参照してください。

2. APB をビルドし、プッシュします。

```
$ apb build
$ apb push
```

3. Web コンソールを使用して APB をプロビジョニングします。

プロビジョニング後に、新規サービスが Web コンソールまたは CLI に表示されます。Web コンソールでは、アプリケーションの **Overview** ページの **Networking** の下か、または **Applications → Services** の下で新規サービスをクリックできます。負荷が分散されたアプリケーションに使用するために使用できるサービスの IP アドレスが表示されます。

コマンドラインでサービスの情報を表示するには、以下を実行できます。

```
$ oc project getting-started
$ oc get services
$ oc describe services/my-test
```

describe は、サービスにアクセスするために使用する IP アドレスを表示します。ただし、ユーザーがアプリケーションにアクセスするために IP アドレスを使用する方法は通常は理想的な方法ではありません。その代わりに、次のセクションで説明されるように **route** を作成する必要があります。

ヒント

次に進む前にクリーンアップし、再度プロビジョニングを実行するために、**getting-started** プロジェクトを削除してから再作成するか、または新規のプロジェクトを作成することができます。

3.1.4.1.3. ルートの作成

アプリケーションへの外部アクセスは、信頼できる名前付きルートで公開できます。

1. **provision-my-test-apb/tasks/main.yml** ファイルを変更して以下を追加します。

```
- name: create my-test route
  openshift_v1_route:
    name: my-test
    namespace: '{{ namespace }}'
    labels:
      app: my-test
      service: my-test
    to_name: my-test
    spec_port_target_port: web
```

to_name はターゲットサービスの名前です。**spec_port_target_port** はターゲットサービスのポートの名前を参照します。詳細は、「**openshift_v1_route** モジュール」を参照してください。

2. APB をビルドし、プッシュします。

```
$ apb build
```

```
$ apb push
```

3. Web コンソールを使用して APB をプロビジョニングします。

プロビジョニング後に、作成される新規ルートが表示されます。Web コンソールの **getting-started** プロジェクトの **Overview** ページでは、アクティブでクリック可能なルートリンクがアプリケーションに一覧表示されます。ルートをクリックするか、または URL にアクセスすると、**hello-world** アプリケーションが起動します。

CLI からルート情報を表示することもできます。

```
$ oc project getting-started

$ oc get routes
NAME      HOST/PORT                                   PATH      SERVICES
PORT      TERMINATION   WILDCARD
my-test   my-test-getting-started.172.17.0.1.nip.io             my-test
web                     None

$ oc describe routes/my-test
Name:   my-test
Namespace:  getting-started
...
```

この時点で、**my-test** アプリケーションは完全に機能し、負荷分散が行われており、拡張可能かつアクセス可能です。完成した APB を **hello-world-apb** サンプルリポジトリーの**hello-world** APB と比較できます。

3.1.4.2. プロビジョニング解除

プロビジョニング解除タスクについては、すべてのプロビジョニングされたリソースを、通常は作成時と反対の順序で破棄する必要があります。

プロビジョニング解除アクションを追加するには、**deprovision.yml** ファイルが **playbooks/** ディレクトリーの下にあり、関連タスクが **roles/deprovision-my-test-apb/tasks/main.yml** になければなりません。これらのファイルはどちらもすでに作成されている必要があります。

```
my-test-apb/
├── apb.yml
├── Dockerfile
├── playbooks
│   └── provision.yml (1)
└── roles
    └── deprovision-my-test-apb
        └── tasks
            └── main.yml (2)
```

(1) このファイルを検査します。

(2) このファイルを編集します。

deprovision.yml ファイルの内容は、異なるロールを呼び出すことを除くとプロビジョニングタスクと同様です。

playbooks/deprovision.yml

```
- name: my-test-apb playbook to deprovision the application
  hosts: localhost
  gather_facts: false
  connection: local
  roles:
  - role: ansible.kubernetes-modules
    install_python_requirements: no
  - role: ansibleplaybookbundle.asb-modules
  - role: deprovision-my-test-apb
    playbook_debug: false
```

このロールをファイル **roles/deprovision-my-test-apb/tasks/main.yml** で編集します。タスクのコメントを解除することにより、コメントなしの生成されるファイルは以下のようになります。

```
- openshift_v1_route:
    name: my-test
    namespace: '{{ namespace }}'
    state: absent

- k8s_v1_service:
    name: my-test
    namespace: '{{ namespace }}'
    state: absent

- openshift_v1_deployment_config:
    name: my-test
    namespace: '{{ namespace }}'
    state: absent
```

先に作成された **provision.yml** ファイルでは、デプロイメント設定、サービス、次にルートを作成しました。プロビジョンニング解除アクションの場合、リソースを逆の順序で削除する必要があります。これは、リソースを **namespace** および **name** で識別し、これを **`state: absent`** とマークして実行できます。

プロビジョニング解除テンプレートを実行するには、**Deployed Services** 一覧のメニューをクリックし、**Delete** を選択します。

3.1.4.2.1. バインド

先のセクションでは、スタンドアロンアプリケーションをデプロイする方法を説明しました。しかし、ほとんどの場合、アプリケーションは他のアプリケーションと通信し、とくにデータソースと通信する必要があります。以下のセクションでは、**my-test-apb** からデプロイされた **hello-world** アプリケーションが使用できる PostSQL データベースを作成します。

3.1.4.2.1.1. 準備

正常に開始できるように、PostgreSQL をプロビジョニングおよびプロビジョニング解除するために必要なファイルを作成します。

注記

詳細な例については、PostgreSQL example APB を参照してください。

1. **--bindable** オプションを使用して APB を初期化します。

```
$ apb init my-pg-apb --bindable
```

これは、いくつかの違いはあるものの、通常の APB ファイル構造を作成します。

```
my-pg-apb/
├── apb.yml (1)
├── Dockerfile
├── playbooks
│   ├── bind.yml (2)
│   ├── deprovision.yml
│   ├── provision.yml
│   └── unbind.yml (3)
└── roles
    ├── bind-my-pg-apb
    │   └── tasks
    │       └── main.yml (4)
    ├── deprovision-my-pg-apb
    │   └── tasks
    │       └── main.yml
    ├── provision-my-pg-apb
    │   └── tasks
    │       └── main.yml (5)
    └── unbind-my-pg-apb
        └── tasks
            └── main.yml (6)
```

(1) **bindable** フラグは **true** に設定されます。

(2) 新規ファイル

(3) 新規ファイル

(4) 新規の空ファイル

(5) エンコードされたバインド認証情報

(6) 新規の空ファイル

通常ファイルのほかに、新規 Playbook **bind.yml**、**unbind.yml**、およびそれらの関連付けられたロールが表示されます。**bind.yml** および **unbind.yml** ファイルはどちらも空であり、デフォルトのバインド動作を使用しているために空のままになります。

2. **apb.yml** ファイルを編集します。**bindable: true** の設定に注意してください。それらの変更のほかにも、PostgreSQL を設定するためにいくつかのパラメーターを **apb.yml** に追加する必要があります。それらは、新規 APB のプロビジョニング時の Web コンソールでの選択可能なフィールドになります。

```
version: 1.0
name: my-pg-apb
description: This is a sample application generated by apb init
bindable: True
async: optional
```

```
metadata:
  displayName: my-pg
plans:
  - name: default
    description: This default plan deploys my-pg-apb
    free: True
    metadata: {}
    # edit the parameters and add the ones below.
    parameters:
      - name: postgresql_database
        title: PostgreSQL Database Name
        type: string
        default: admin
      - name: postgresql_user
        title: PostgreSQL User
        type: string
        default: admin
      - name: postgresql_password
        title: PostgreSQL Password
        type: string
        default: admin
```

playbooks/provision.yml は以下のようになります。

```
- name: my-pg-apb playbook to provision the application
  hosts: localhost
  gather_facts: false
  connection: local
  roles:
  - role: ansible.kubernetes-modules
    install_python_requirements: no
  - role: ansibleplaybookbundle.asb-modules
  - role: provision-my-pg-apb
    playbook_debug: false
```

playbooks/deprovision.yml は以下のようになります。

```
- name: my-pg-apb playbook to deprovision the application
  hosts: localhost
  gather_facts: false
  connection: local
  roles:
  - role: ansible.kubernetes-modules
    install_python_requirements: no
  - role: deprovision-my-pg-apb
    playbook_debug: false
```

3. **roles/provision-my-pg-apb/tasks/main.yml** ファイルを編集します。このファイルは、多くの点で **hello-world** アプリケーションをミラーリングしますが、再起動間のデータを保存し、デプロイメント設定の各種設定オプションを保存するために 永続ボリューム (PV) を追加します。
さらに、新規タスクがプロビジョニングタスク後に下部に追加されます。プロビジョニングプロセスで作成された認証情報を保存するには、OAB で取得できるようにそれらをエンコードする必要があります。モジュール **asb_encode_binding** を使用する新規タスクでこれを実行します。

このファイルのすべての内容を安全に削除し、これを以下に置き換えることができます。

```
# New persistent volume claim
- name: create volumes
  k8s_v1_persistent_volume_claim:
    name: my-pg
    namespace: '{{ namespace }}'
    state: present
    access_modes:
      - ReadWriteOnce
    resources_requests:
      storage: 1Gi

- name: create deployment config
  openshift_v1_deployment_config:
    name: my-pg
    namespace: '{{ namespace }}'
    labels:
      app: my-pg
      service: my-pg
    replicas: 1
    selector:
      app: my-pg
      service: my-pg
    spec_template_metadata_labels:
      app: my-pg
      service: my-pg
    containers:
    - env:
      - name: POSTGRESQL_PASSWORD
        value: '{{ postgresql_password }}'
      - name: POSTGRESQL_USER
        value: '{{ postgresql_user }}'
      - name: POSTGRESQL_DATABASE
        value: '{{ postgresql_database }}'
      image: docker.io/centos/postgresql-94-centos7
      name: my-pg
      ports:
      - container_port: 5432
        protocol: TCP
      termination_message_path: /dev/termination-log
      volume_mounts:
      - mount_path: /var/lib/pgsql/data
        name: my-pg
      working_dir: /
    volumes:
    - name: my-pg
      persistent_volume_claim:
        claim_name: my-pg
      test: false
      triggers:
      - type: ConfigChange

- name: create service
  k8s_v1_service:
    name: my-pg
```

```
    namespace: '{{ namespace }}'
    state: present
    labels:
      app: my-pg
      service: my-pg
    selector:
      app: my-pg
      service: my-pg
    ports:
    - name: port-5432
      port: 5432
      protocol: TCP
      target_port: 5432

# New encoding task makes credentials available to future bind
operations
- name: encode bind credentials
  asb_encode_binding:
    fields:
      DB_TYPE: postgres
      DB_HOST: my-pg
      DB_PORT: "5432"
      DB_USER: "{{ postgresql_user }}"
      DB_PASSWORD: "{{ postgresql_password }}"
      DB_NAME: "{{ postgresql_database }}"
```

encode bind credentials タスクは利用可能なフィールドを環境変数として利用可能にします。これには、**DB_TYPE**、**DB_HOST**、**DB_PORT**、**DB_USER**、**DB_PASSWORD**、および **DB_NAME** が含まれます。これは、**bind.yml** ファイルが空のままの場合のデフォルト動作になります。すべてのアプリケーション (**hello-world** など) がこれらの環境変数を使用でき、バインド操作後に設定済みのデータベースに接続できます。

4. **roles/deprovision-my-pg-apb/tasks/main.yml** を編集し、以下の行のコメントを解除して、作成されたリソースがプロビジョニング解除時に削除されるようにします。

```
- k8s_v1_service:
    name: my-pg
    namespace: '{{ namespace }}'
    state: absent

- openshift_v1_deployment_config:
    name: my-pg
    namespace: '{{ namespace }}'
    state: absent

- k8s_v1_persistent_volume_claim:
    name: my-pg
    namespace: '{{ namespace }}'
    state: absent
```

5. 最後に APB をビルドし、プッシュします。

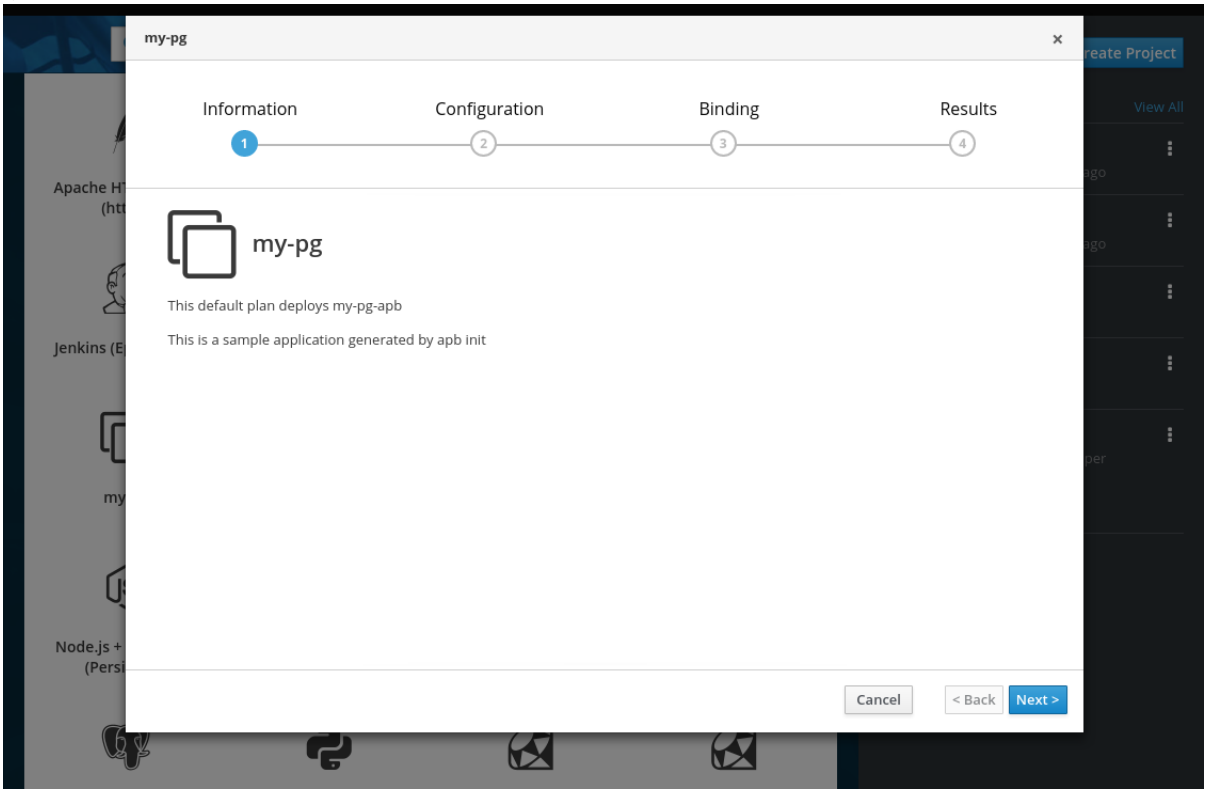
```
$ apb build
$ apb push
```

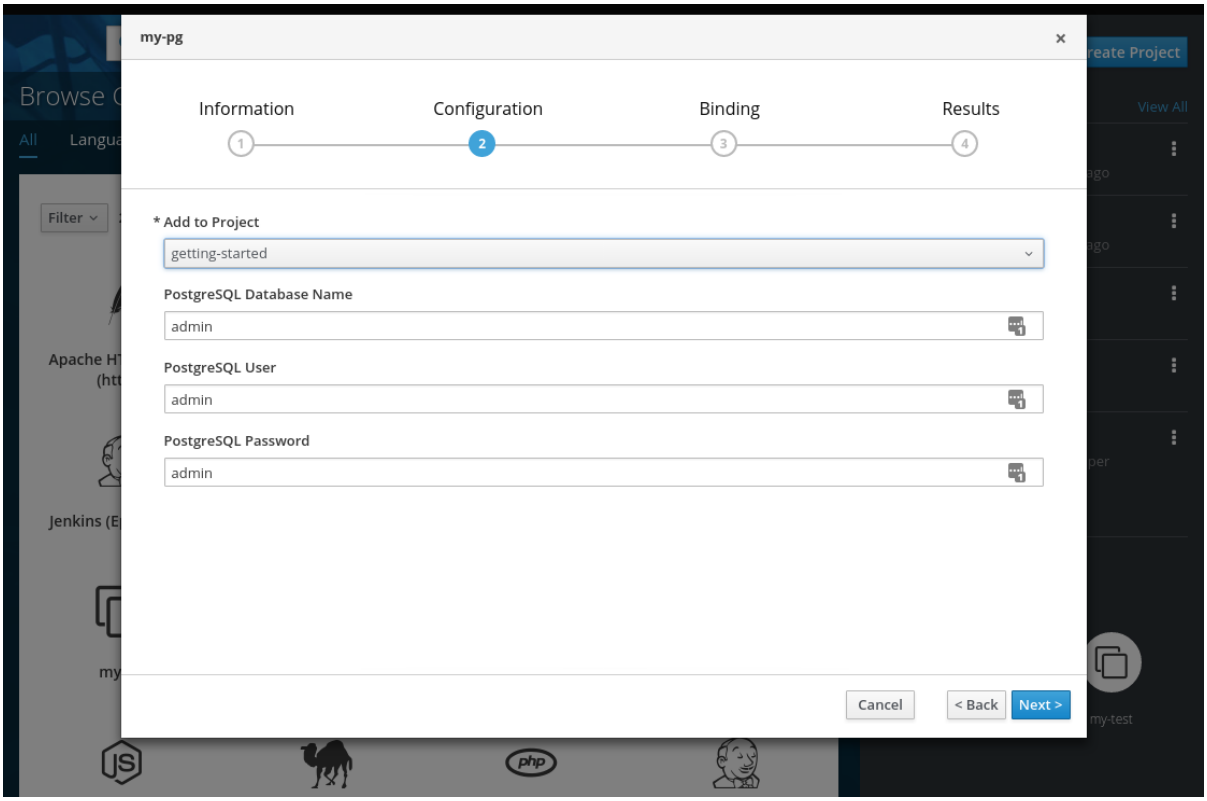
この時点で、APB は完全に機能する PostgreSQL データベースをクラスターに作成できます。これについては、次のセクションでテストできます。

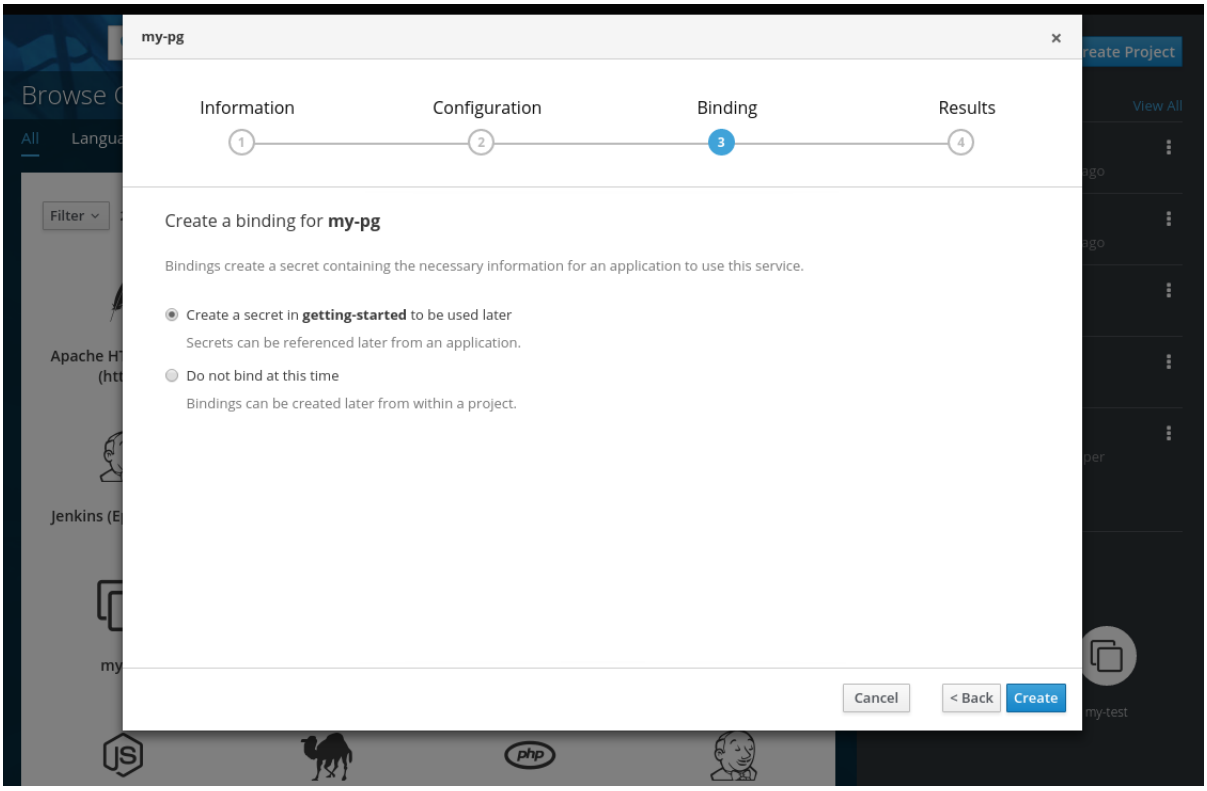
3.1.4.2.1.2. UI からの実行

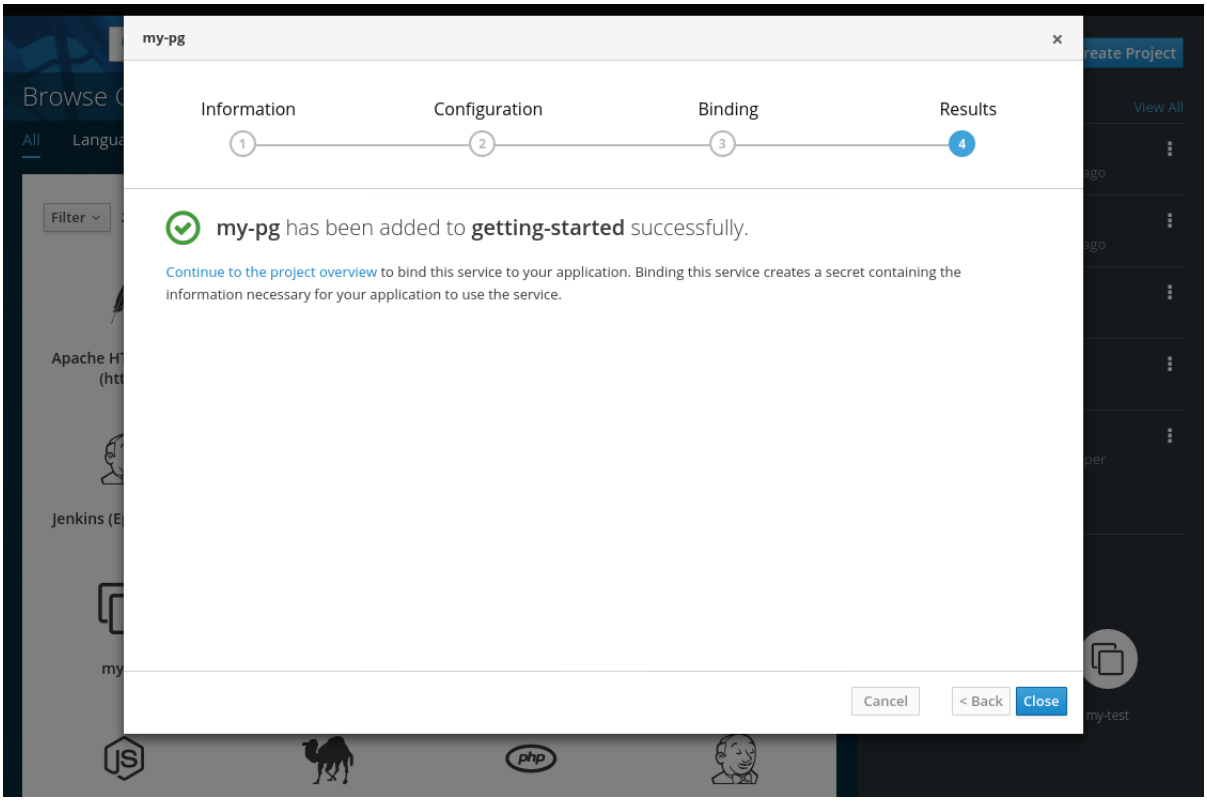
アプリケーションをテストするには、**hello-world** アプリケーションをプロビジョニングされた PostgreSQL データベースにバインドできます。このチュートリアルの「プロビジョニング」セクションで事前に作成されたアプリケーションを使用するか、または **hello-world-apb** を使用できます。

1. まず、**my-test-apb** をプロビジョニングします。
2. 次に、**my-pg-apb** をプロビジョニングし、**シークレットを作成する** オプションを選択します。



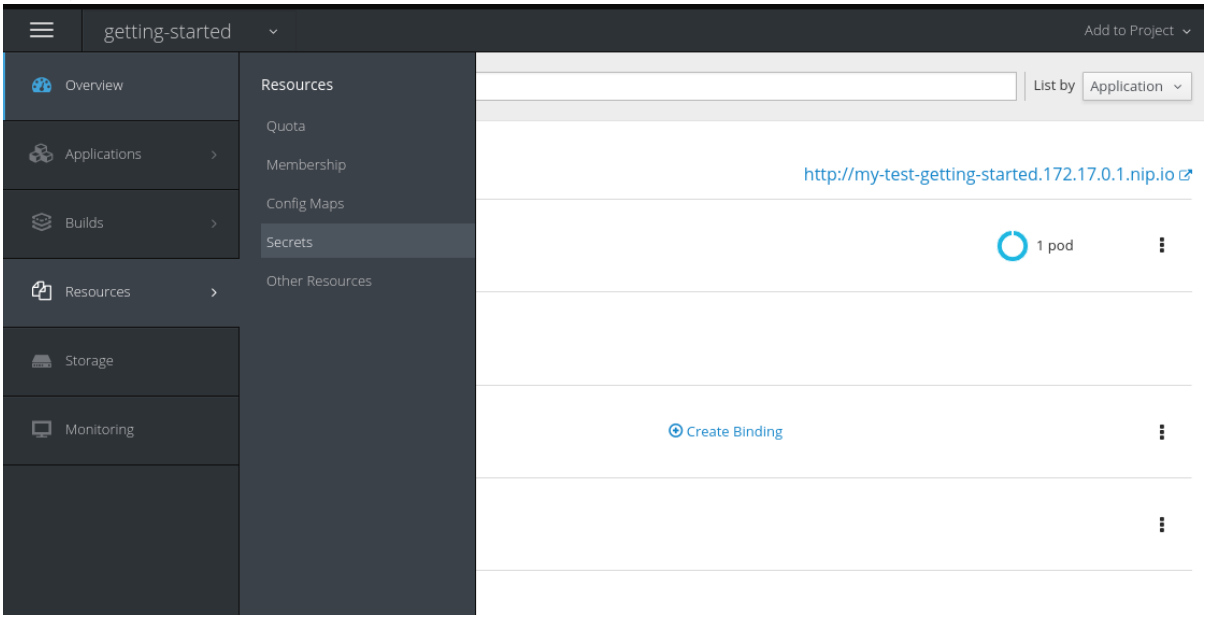

my-pg
Information
Configuration
Binding
Results
1
2
3
4
* Add to Project
getting-started
PostgreSQL Database Name
admin
PostgreSQL User
admin
PostgreSQL Password
admin
Cancel
< Back
Next >

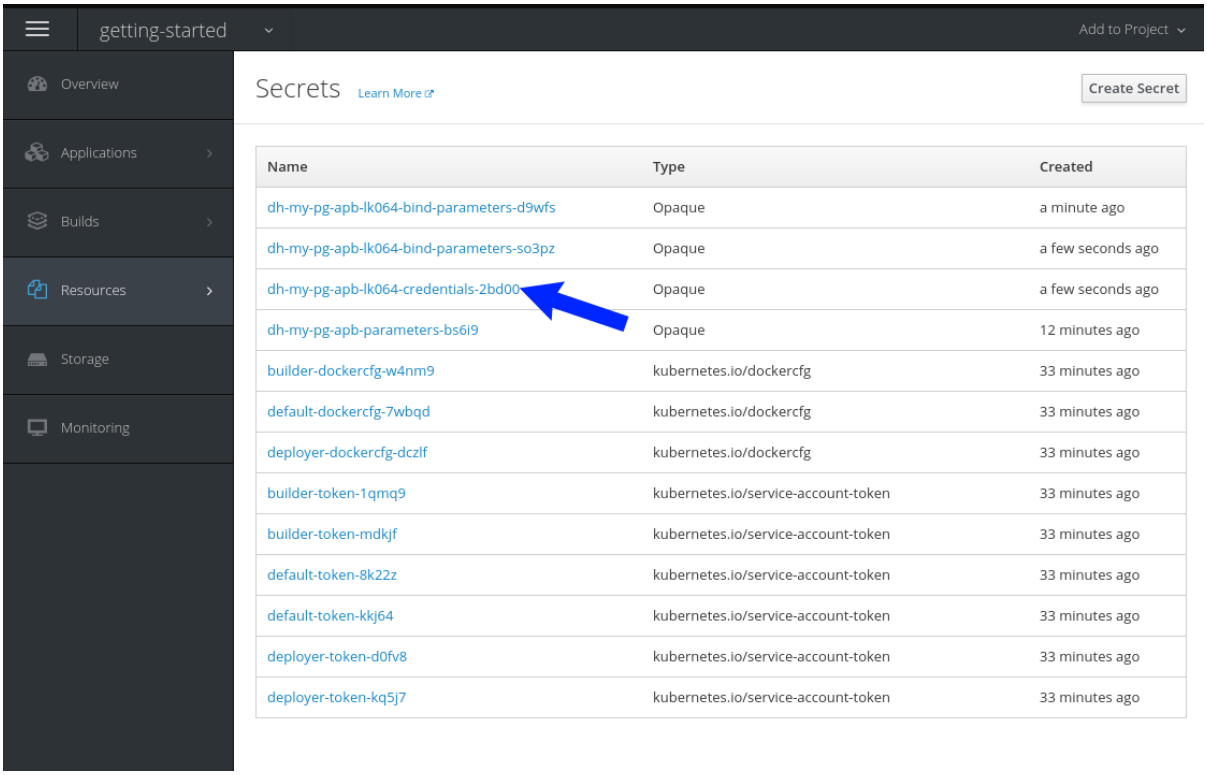

my-pg
Information
Configuration
Binding
Results
1
2
3
4
Create a binding for my-pg
Bindings create a secret containing the necessary information for an application to use this service.
Create a secret in getting-started to be used later
Secrets can be referenced later from an application.
Do not bind at this time
Bindings can be created later from within a project.
Cancel
< Back
Create



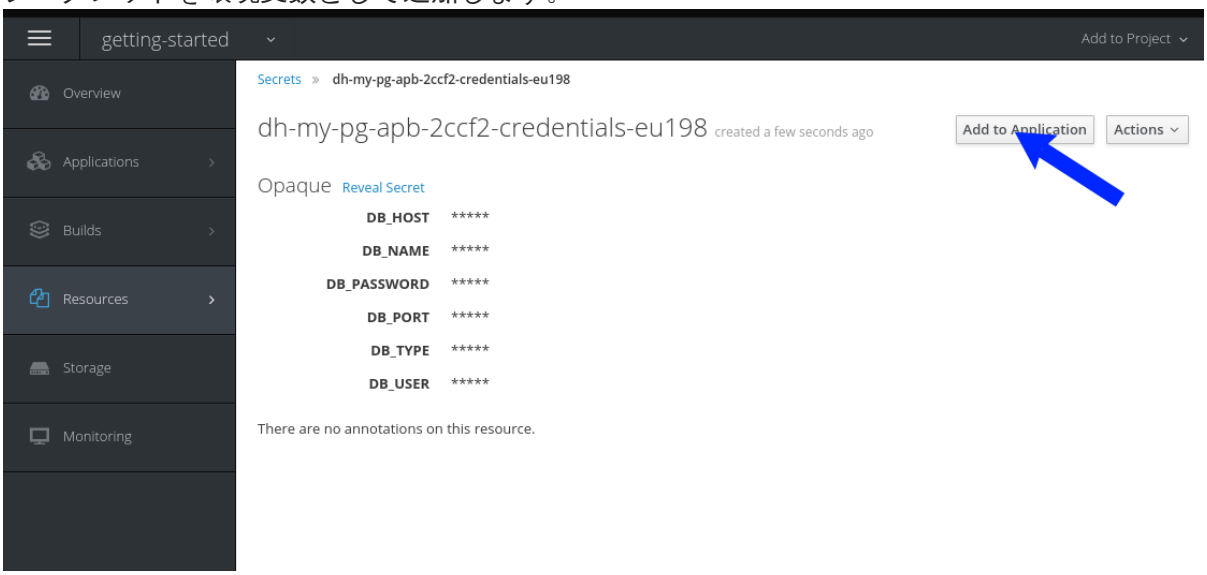
3. 次に、プロジェクトに移動します (まだの場合)。**hello-world** アプリケーションと PostgreSQL データベースの両方を確認できます。プロビジョニング時にバインディングを作成することを選択しなかった場合は、ここで **バインディングの作成** リンクを使用してこれを実行できます。

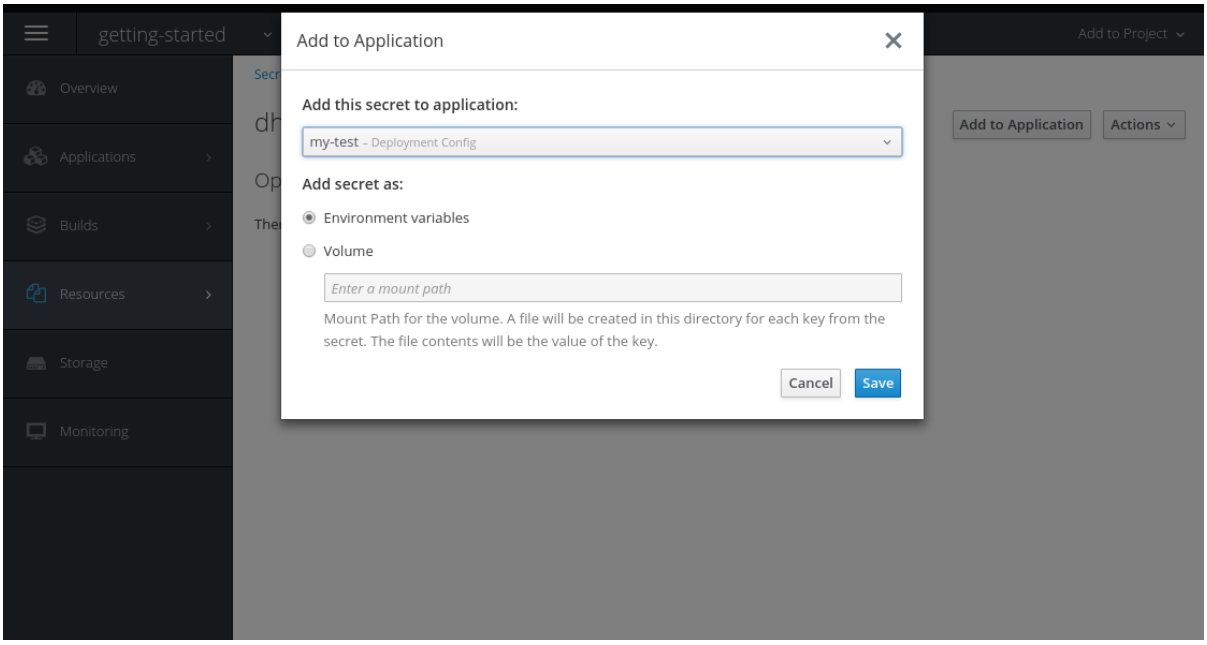
4. バインディングが作成されたら、バインディングで作成されたシークレットをアプリケーションに追加する必要があります。まず、「**Resources → Secrets**」ページでシークレットに移動します。



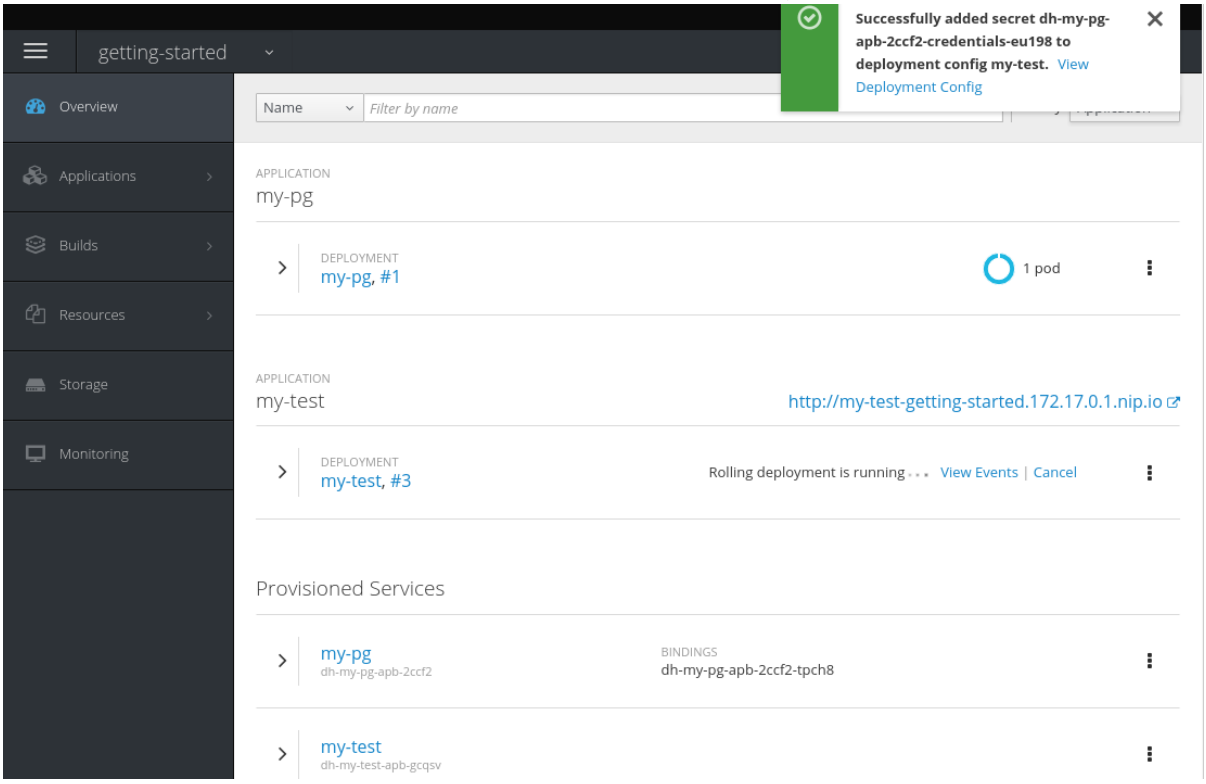


5. シークレットを環境変数として追加します。

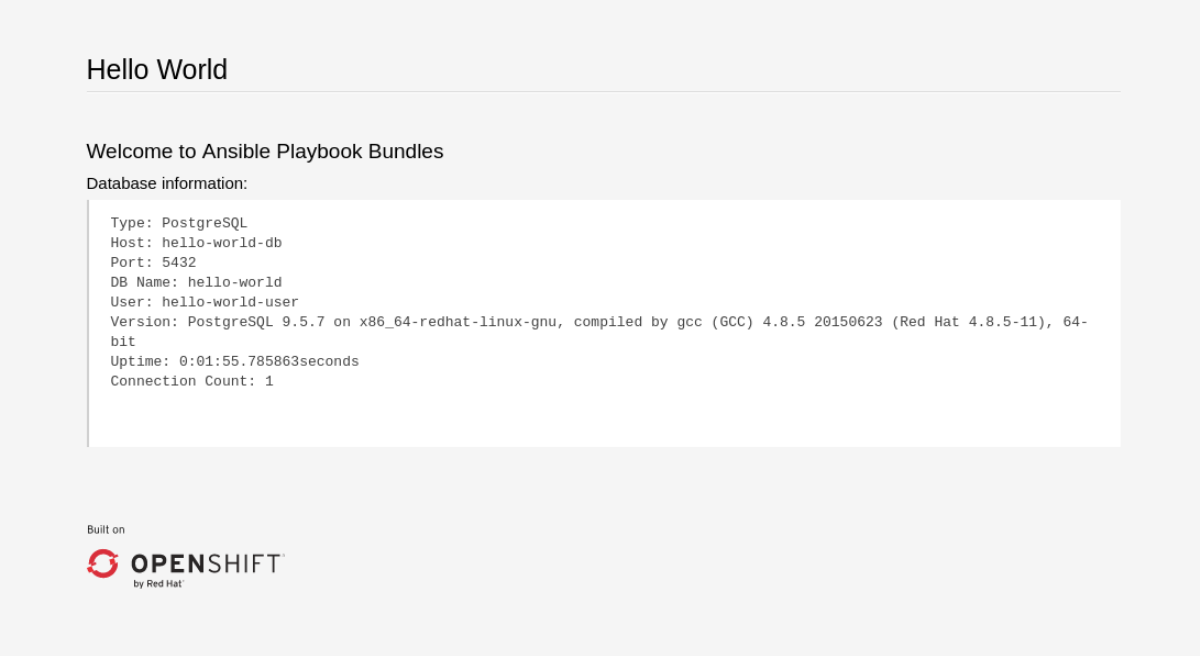




6. これを追加したら、**Overview** ページに戻ることができます。**my-test** アプリケーションは依然として設定変更による再デプロイの実行中である可能性があります。その場合には、ルートをクリックしてアプリケーションを表示できるまで待機します。



ルートのクリック後に、**hello-world** アプリケーションが **my-pg** データベースを検出し、これに接続されていることを確認できます。



3.1.4.2.2. テスト

テストアクションは、APB がサービスカタログに公開される前に基本的なサニティーチェックをパスするものであることを確認することを目的としています。これらはライブサービスをテストするように意図されていません。OpenShift Container Platform には、プロビジョニング時に追加できる liveness および readiness プローブを使用してライブサービスをテストする機能があります。

テストの実際の実装については、APB の作成者が決定することができます。以下のセクションでは、これについてのガイダンスおよびベストプラクティスを説明します。

3.1.4.2.2.1. テストアクションの作成

APB のテストアクションを作成するには、以下を実行します。

- **playbooks/test.yml** ファイルを組み込みます。
- テストのデフォルトを **playbooks/vars/** ディレクトリーに組み込みます。

```
my-apb/
├── ...
├── playbooks/
    ├── test.yml
    └── vars/
        └── test_defaults.yml
```

APB のテストのオーケストレーションを実行するには、**test.yml** ファイルの include_vars および include_role モジュールを使用する必要があります。

test.yml

```
- name: test media wiki abp
  hosts: localhost
  gather_facts: false
  connection: local

  roles:
```

```
  - role: ansible.kubernetes-modules (1)
    install_python_requirements: no

  post_tasks:
  - name: Load default variables for testing (2)
    include_vars: test_defaults.yaml
  - name: create project for namespace
    openshift_v1_project:
      name: '{{ namespace }}'
  - name: Run the provision role. (3)
    include_role:
      name: provision-mediawiki123-apb
  - name: Run the verify role. (4)
    include_role:
      name: verify-mediawiki123-apb
```

(1) Ansible Kubernetes モジュールを読み込みます。

(2) テストロールからプロビジョニングするために必要なデフォルト値を組み込みます。

(3) 実行するプロビジョニングロールを組み込みます。

(4) 実行する検証ロールを組み込みます。「検証ロールの作成」を参照してください。

3.1.4.2.2.2. 検証ロールの作成

検証ロール では、プロビジョニングが失敗したか、または成功したかを判別できます。**verify_<name>** ロールは **roles/** ディレクトリーに配置されます。これは通常の Ansible ロールです。

```
my-apb/
├── ...
└── roles/
    ├── ...
    └── verify_<name>
        ├── defaults
        │   └── defaults.yml
        └── tasks
            └── main.yml
```

main.yml ファイルのタスクサンプルは以下のようになります。

```
 - name: url check for media wiki
   uri:
     url: "http://{{ route.route.spec.host }}"
     return_content: yes
   register: webpage
   failed_when: webpage.status != 200
```

3.1.4.2.2.3. テスト結果の保存

また、**asb_save_test_result** モジュールを検証ロールで使用することもできます。これにより、APB はテスト結果を保存し、**apb test** コマンドはそれらを返すことができるようになります。APB Pod はツールがテスト結果を取得するまで有効な状態のままになります。

たとえば、**asb_save_test_result** の使用を先の **main.yml** のサンプルに追加します。

```
  - name: url check for media wiki
    uri:
      url: "http://{{ route.route.spec.host }}"
      return_content: yes
    register: webpage

  - name: Save failure for the web page
    asb_save_test_result:
      fail: true
      msg: "Could not reach route and retrieve a 200 status code. Recieved
status - {{ webpage.status }}"
    when: webpage.status != 200

  - fail:
      msg: "Could not reach route and retrieve a 200 status code. Recieved
status - {{ webpage.status }}"
    when: webpage.status != 200

  - name: Save test pass
    asb_save_test_result:
      fail: false
    when: webpage.status == 200
```

3.1.4.2.2.4. テストアクションの実行

テストアクションを定義した後に、CLI ツールを使用してテストを実行できます。

```
$ apb test
```

テストアクションは以下を実行します。

- イメージのビルド
- サービスブローカーで実行を開始される場合と同様の Pod の起動
- 保存されている場合、テスト結果の取得

実行が終了した後の Pod のステータスはテストのステータスを判別するものになります。Pod がエラー状態の場合は障害が生じたことを示し、コマンドはテストが不成功であったことを報告します。

3.2. APB の作成: 参照

3.2.1. 概要

「使用開始」のトピックでは初回の Ansible Playbook Bundle (APB) の作成を手順ごとに説明していますが、このトピックでは、詳細な参考情報を提供します。ここでは APB を構成する基本的なコンポーネントについて詳細に説明し、経験豊かな APB 開発者が APB 内の個々のコンポーネントをよよく理解するのに役立つ情報を提供します。

完成した APB サンプルについては、GitHub の **ansibleplaybookbundle** 組織で APB を参照できます。

3.2.2. ディレクトリー構造

以下は、APB のディレクトリー構造のサンプルを示しています。

```
example-apb/
├── Dockerfile
├── apb.yml
└── roles/
|   └── example-apb-openshift
|       ├── defaults
|       |   └── main.yml
|       └── tasks
|           └── main.yml
└── playbooks/
    └── provision.yml
    └── deprovision.yml
    └── bind.yml
    └── unbind.yml
```

3.2.3. APB 仕様ファイル

APB 仕様ファイルは **apb.yml** に置かれ、ここではアプリケーションの概要情報が宣言されます。以下は APB 仕様の例です。

```
  version: 1.0
  name: example-apb
  description: A short description of what this APB does
  bindable: True
  async: optional (1)
  metadata:
    documentationUrl: <link_to_documentation>
    imageUrl: <link_to_url_of_image>
    dependencies: ['<registry>/<organization>/<dependency_name_1>',
'<registry>/<organization>/<dependency_name_2>']
    displayName: Example App (APB)
    longDescription: A longer description of what this APB does
    providerDisplayName: "Red Hat, Inc."
  plans:
    - name: default
      description: A short description of what this plan does
      free: true
      metadata:
        displayName: Default
        longDescription: A longer description of what this plan deploys
        cost: $0.00
      parameters:
        - name: parameter_one
          required: true
          default: foo_string
          type: string
          title: Parameter One
          maxlength: 63
        - name: parameter_two
```

```
          required: true
          default: true
          title: Parameter Two
          type: boolean
```

1 非同期バインドおよびバインド解除は実験的な機能であり、デフォルトではサポートされておらず、有効にされていません。

3.2.3.1. 上部の構造

フィールド	説明
version	APB 仕様のバージョンです。詳細は、「APB 仕様のバージョン管理」を参照してください。
name	APB の名前です。名前は有効な ASCII でなければならず、小文字、数字、アンダースコア、ピリオド、およびダッシュを含めることができます。有効なタグ名については、Docker のガイドラインを参照してください。
description	この APB の簡単な説明です。
bindable	この APB をバインド可能にするかについてのブール値オプションです。許可されるフィールドは **true** または **false** です。
metadata	関連するメタデータ情報を宣言するディクショナリーフィールドです。
plans	デプロイできるプランの一覧です。詳細は、「プラン」を参照してください。

3.2.3.2. メタデータ

フィールド	説明
documentationUrl	アプリケーションのドキュメントへの URL です。
imageUrl	サービスカタログについての Web コンソールに表示されるイメージの URL です。
dependencies	APB 内で使用できるイメージの一覧です。
displayName	Web コンソールに表示されるこの APB の名前です。

フィールド	説明
longDescription	APB が Web コンソールでクリックされる際に表示される詳細の説明です。
providerDisplayName	この APB を使用できるように提供しているプロバイダーの名前です。

3.2.3.3. プラン

プランは一覧として宣言されます。このセクションでは、プランの各フィールドの説明内容について説明します。

フィールド	説明
name	デプロイするプランの任意の名前です。これは APB がサービスカタログからクリックされる際に表示されます。
description	このプランからデプロイされる内容の簡単な説明です。
free	この計画にコストが発生しないかどうかを判別するためのブール値フィールドです。許可されるフィールドは **true** または **false** になります。
metadata	関連するプランメタデータ情報を宣言するディクショナリーフィールドです。詳細は、「プランメタデータ」を参照してください。
parameters	APB への入力として使用されるパラメーターディクショナリーの一覧です。詳細は、「パラメーター」を参照してください。

3.2.3.4. プランメタデータ

フィールド	説明
displayName	Web コンソールに表示されるプランの名前です。
longDescription	このプランがデプロイする内容の詳細な説明です。
cost	プランのデプロイにかかるコストです。許可されるフィールドは **$x.yz** です。

3.2.3.5. パラメーター

parameters セクションの各項目には複数のフィールドがある場合があります。**name** フィールドは必

須です。パラメーターの順序は、OpenShift Container Platform Web コンソールのフォームの順序で表示されます。

```
parameters:
  - name: my_param
    title: My Parameter
    type: enum
    enum: ['X', 'Y', 'Z']
    required: True
    default: X
    display_type: select
    display_group: Group 1
```

フィールド	説明
name	APB に渡されるパラメーターの固有の名前です。
title	Web コンソールに表示されるラベルです。
type	リンク json-schema で指定されるパラメーターのデータタイプです (**string**、**number**、**int**、**boolean**、または **enum**)。Web コンソールのデフォルトの入力フィールドタイプは、**display_type** が割り当てられていない場合に割り当てられます。
required	パラメーターが APB の実行に必須かどうかを示します。Web コンソールの必須フィールドです。
default	パラメーターに割り当てられるデフォルト値です。
display_type	Web コンソールの表示タイプです。たとえば、**password** として入力された文字列を上書きして、これを Web コンソールで非表示にできます。許可されるフィールドには、**text**、**textarea**、**password**、**checkbox**、または **select** が含まれます。
display_group	パラメーターが一致する **display_group** フィールドを持つ隣接パラメーターと共にグループで表示されます。上記の例では、別のフィールドを **display_group: Group 1** の下に追加すると、Web コンソールの見出し **Group 1** の下にそれらがグループとして表示されます。

パラメーターの詳細の一覧を使用する場合、共有パラメーター一覧を使用すると便利です。この例として、rhscl-postgresql-apb を参照してください。

3.2.3.6. APB 仕様のバージョン管理

APB 仕様は **x.y** 形式のセマンティックバージョニングを使用します。ここで、**x** はメジャーリリースで、**y** はマイナーリリースです。

現行の仕様バージョンは **1.0** です。

3.2.3.6.1. メジャーバージョン

APB 仕様は、API の互換性を失う変更が仕様に導入されるたびにメジャーバージョンを増分します。その例には以下が含まれます。

- 必須フィールドの導入または削除。
- YAML 形式の変更。
- 新機能。

3.2.3.6.2. マイナーバージョン

APB 仕様は、API の互換性に影響がない変更が仕様に導入されるたびにマイナーバージョンを増分します。その例には以下が含まれます。

- オプションフィールドの導入または削除。
- スペルの修正。
- 新規オプションの既存フィールドへの導入。

3.2.4. Dockerfile

Dockerfile は APB イメージを実際にビルドする際に使用されます。そのため、これを独自のニーズに合わせてカスタマイズする必要があります。たとえば、PostgreSQL との対話が必要な Playbook を実行する場合、**yum install** コマンドを追加して必要なパッケージをインストールすることが必要になる可能性があります。

```
FROM ansibleplaybookbundle/apb-base
MAINTAINER Ansible Playbook Bundle Community

LABEL "com.redhat.apb.spec"=\
"<------------base64-encoded-spec------------>"


COPY roles /opt/ansible/roles
COPY playbooks /opt/apb/actions
RUN chmod -R g=u /opt/{ansible,apb}


### INSTALL THE REQUIRED PACKAGES
RUN yum -y install python-boto postgresql && yum clean all

USER apb
```

3.2.5. APB アクション (Playbook)

APB のアクションは APB の実行で使用されるコマンドです。サポートされている標準的なアクションは以下になります。

- provision (プロビジョニング)
- deprovision (プロビジョニング解除)
- bind (バインド)
- unbind (バインド解除)
- test (テスト)

アクションを有効にするには、**playbooks/** ディレクトリーに **<action>.yml** という名前の有効なファイルがなければなりません。これらの Playbook はすべてのことを実行できる、つまり、希望するすべてのアクションを作成できます。たとえば、mediawiki-apb には **update** アクションを作成する Playbook があります。

ほとんどの APB にはリソースを作成するためのプロビジョニングアクションと、サービスを削除する際にリソースを破棄するためのプロビジョニング解除アクションがあります。

バインドおよびバインド解除アクションは、1 つのサービスの位置情報 (coordinate) を別のサービスで利用可能にする必要がある場合に使用されます。通常、これはデータサービスを作成し、これをアプリケーションで利用可能にする場合に該当します。現時点では、この位置情報 (coordinate) はプロビジョニング時に利用可能になります。

位置情報 (coordinate) を別のサービスで適切に利用可能にするには、**asb_encode_binding** モジュールを使用します。このモジュールは APB のプロビジョニングロールの最後に呼び出され、これは OpenShift Ansible Broker (OAB) にバインド認証情報を返します。

```
- name: encode bind credentials
  asb_encode_binding:
    fields:
      EXAMPLE_FIELD: foo
      EXAMPLE_FIELD2: foo2
```

3.2.6. 共通リソースの使用

このセクションでは、APB の開発時に作成される共通の OpenShift Container Platform リソースの一覧について説明します。利用可能なリソースモジュールの詳細の一覧については、「Ansible Kubernetes Module」を参照してください。

3.2.6.1. サービス

以下は、**hello-world** という名前のサービスを作成するためのサンプルの Ansible タスクです。APB の **namespace** 変数は Web コンソールから起動される場合に OAB によって提供されます。

プロビジョニング

```
- name: create hello-world service
  k8s_v1_service:
    name: hello-world
    namespace: '{{ namespace }}'
    labels:
      app: hello-world
      service: hello-world
    selector:
      app: hello-world
```

```
      service: hello-world
    ports:
      - name: web
        port: 8080
        target_port: 8080
```

プロビジョニング解除

```
- k8s_v1_service:
    name: hello-world
    namespace: '{{ namespace }}'
    state: absent
```

3.2.6.2. デプロイメント設定

以下は、サービス **hello-world** にマップするイメージ **docker.io/ansibleplaybookbundle/hello-world** のデプロイメント設定を作成するサンプルの **Ansible** タスクです。

プロビジョニング

```
- name: create deployment config
  openshift_v1_deployment_config:
    name: hello-world
    namespace: '{{ namespace }}'
    labels:
      app: hello-world
      service: hello-world
    replicas: 1
    selector:
      app: hello-world
      service: hello-world
    spec_template_metadata_labels:
      app: hello-world
      service: hello-world
    containers:
    - env:
      image: docker.io/ansibleplaybookbundle/hello-world:latest
      name: hello-world
      ports:
      - container_port: 8080
        protocol: TCP
```

プロビジョニング解除

```
- openshift_v1_deployment_config:
    name: hello-world
    namespace: '{{ namespace }}'
    state: absent
```

3.2.6.3. ルート

以下は、サービス **hello-world** にマップする **hello-world** という名前のルートを作成する例です。

プロビジョニング

```
- name: create hello-world route
  openshift_v1_route:
    name: hello-world
    namespace: '{{ namespace }}'
    spec_port_target_port: web
    labels:
      app: hello-world
      service: hello-world
    to_name: hello-world
```

プロビジョニング解除

```
- openshift_v1_route:
    name: hello-world
    namespace: '{{ namespace }}'
    state: absent
```

3.2.6.4. 永続ボリューム

以下は、Persistent Volume Claim (PVC, 永続ボリューム要求) リソースとこれを使用するデプロイメント設定を作成する例です。

プロビジョニング

```
# Persistent volume resource
- name: create volume claim
  k8s_v1_persistent_volume_claim:
    name: hello-world-db
    namespace: '{{ namespace }}'
    state: present
    access_modes:
      - ReadWriteOnce
    resources_requests:
      storage: 1Gi
```

リソースに加えて、ボリュームをデプロイメント設定の宣言に追加します。

```
- name: create hello-world-db deployment config
  openshift_v1_deployment_config:
    name: hello-world-db
    ---
    volumes:
    - name: hello-world-db
      persistent_volume_claim:
        claim_name: hello-world-db
      test: false
      triggers:
      - type: ConfigChange
```

プロビジョニング解除

```
- openshift_v1_deployment_config:
    name: hello-world-db
    namespace: '{{ namespace }}'
    state: absent

- k8s_v1_persistent_volume_claim:
    name: hello-world-db
    namespace: '{{ namespace }}'
    state: absent
```

3.2.7. オプション変数

オプションの変数は、環境変数を使用して APB に追加できます。変数を APB に渡すには、**.yml** ファイルで変数置換をエスケープする必要があります。

たとえば、**etherpad-apb** の以下の **roles/provision-etherpad-apb/tasks/main.yml** ファイルについて見てみましょう。

```
- name: create mariadb deployment config
  openshift_v1_deployment_config:
    name: mariadb
    namespace: '{{ namespace }}'
    ...
    - env:
      - name: MYSQL_ROOT_PASSWORD
        value: '{{ mariadb_root_password }}'
      - name: MYSQL_DATABASE
        value: '{{ mariadb_name }}'
      - name: MYSQL_USER
        value: '{{ mariadb_user }}'
      - name: MYSQL_PASSWORD
        value: '{{ mariadb_password }}'
```

APB の変数は **roles/provision-etherpad-apb/defaults/main.yml** ファイルに定義されます。

```
playbook_debug: no
mariadb_root_password: "{{ lookup('env','MYSQL_ROOT_PASSWORD') |
default('admin', true) }}"
mariadb_name: "{{ lookup('env','MYSQL_DATABASE') | default('etherpad',
true) }}"
mariadb_user: "{{ lookup('env','MYSQL_USER') | default('etherpad', true)
}}"
mariadb_password: "{{ lookup('env','MYSQL_PASSWORD') | default('admin',
true) }}"
etherpad_admin_password: "{{ lookup('env','ETHERPAD_ADMIN_PASSWORD') |
default('admin', true) }}"
etherpad_admin_user: "{{ lookup('env','ETHERPAD_ADMIN_USER') |
default('etherpad', true) }}"
etherpad_db_host: "{{ lookup('env','ETHERPAD_DB_HOST') |
default('mariadb', true) }}"
state: present
```

3.2.8. 制限付き SCC の使用

OpenShift Container Platform イメージのビルド時に、アプリケーションを root ユーザーとして実行していないことは重要になります (可能な場合)。**制限付き** SCC で実行される場合、アプリケーションのイメージはランダム UID で起動します。これは、アプリケーションフォルダーが root ユーザーで所有されている場合には問題を生じさせます。

この問題の適切な回避策として、ユーザーを root グループに追加し、アプリケーションを root グループによって所有されるようにすることができます。任意のユーザー ID をサポートする方法についての詳細は、「OpenShift Container Platform-Specific Guidelines」を参照してください。

以下は、**/usr/src** で実行されるノードアプリケーションの**Dockerfile** の例です。このコマンドは、アプリケーションが **/usr/src** にインストールされ、関連付けられた環境変数が設定された後に実行されます。

```
ENV USER_NAME=haste \
    USER_UID=1001 \
    HOME=/usr/src

RUN useradd -u ${USER_UID} -r -g 0 -M -d /usr/src -b /usr/src -s
/sbin/nologin -c "<username> user" ${USER_NAME} \
               && chown -R ${USER_NAME}:0 /usr/src \
               && chmod -R g=u /usr/src /etc/passwd
USER 1001
```

3.2.9. ConfigMap の APB 内での使用

Ansible モジュールのバグ対策として、Ansible から ConfigMaps を作成するために一時的な回避策を使用できます。

ConfigMaps の 1 つの一般的なユースケースとして、APB のパラメーターがアプリケーションまたはサービスの設定ファイル内で使用される場合があります。ConfigMap モジュールにより、ConfigMap を Pod にボリュームとしてマウントできます。これは設定ファイルを保存するために使用できます。この方法では、ConfigMap を APB パラメーターから作成するために Ansible の **テンプレート** モジュール機能を活用することもできます。

以下は、ボリュームとして Pod にマウントされた Jinja テンプレートから ConfigMap を作成する例です。

```
- name: Create hastebin config from template
  template:
    src: config.js.j2
    dest: /tmp/config.js

- name: Create hastebin configmap
  shell: oc create configmap haste-config --from-file=haste-
config=/tmp/config.js

<snip>

- name: create deployment config
  openshift_v1_deployment_config:
    name: hastebin
    namespace: '{{ namespace }}'
    labels:
      app: hastebin
      service: hastebin
```

```
    replicas: 1
    selector:
      app: hastebin
      service: hastebin
    spec_template_metadata_labels:
      app: hastebin
      service: hastebin
    containers:
    - env:
      image: docker.io/dymurray/hastebin:latest
      name: hastebin
      ports:
      - container_port: 7777
        protocol: TCP
      volumeMounts:
        - mountPath: /usr/src/haste-server/config
          name: config
    - env:
      image: docker.io/modularitycontainers/memcached:latest
      name: memcached
      ports:
      - container_port: 11211
        protocol: TCP
    volumes:
      - name: config
        configMap:
          name: haste-config
          items:
            - key: haste-config
              path: config.js
```