



Microsoft Azure 自習書シリーズ

開発テスト効率化と継続的インテグレーション

この自習書では、Microsoft が提供するパブリッククラウドサービスである Visual Studio Team Services を利用し、Git リポジトリと連動してアプリのビルドとテストが自動実行される“継続的インテグレーション”環境を構築する一連の流れを、ハンズオン形式で学習体験します。

発行日 : 2018 年 6 月 20 日

更新履歴

版数	発行日	更新履歴
第 1 版	2017 年 7 月 21 日	初版発行
第 2 版	2017 年 8 月 23 日	UI の変更に対応
第 3 版	2017 年 9 月 20 日	SQL Database の作成手順を追加、UI の変更に対応
第 4 版	2017 年 10 月 24 日	UI の変更に対応
第 5 版	2017 年 11 月 27 日	UI の変更に対応
第 6 版	2017 年 12 月 22 日	UI の変更に対応
第 7 版	2018 年 1 月 25 日	UI の変更に対応
第 8 版	2018 年 2 月 16 日	UI の変更に対応
第 9 版	2018 年 3 月 23 日	UI の変更に対応
第 10 版	2018 年 4 月 20 日	UI の変更に対応
第 11 版	2018 年 5 月 28 日	UI の変更に対応
第 12 版	2018 年 6 月 20 日	UI の変更に対応

目次


1. はじめに.....	4
STEP 1. 環境整備.....	5
2. 実習環境の準備.....	6
3. サンプルアプリについて	8
4. Microsoft Azure サブスクリプションの準備	9
5. Visual Studio Team Services へのサインアップ	10
6. Git のインストール.....	14
7. Visual Studio Code のインストール	18
8. Azure App Service Web App の作成.....	24
9. Azure SQL Database の作成.....	30
STEP 2. タスク管理.....	40
10. カンバンボードでのタスク管理.....	41
11. Feature と Backlog Item の追加	42
12. Task の追加.....	46
13. Sprint の開始.....	49
STEP 3. 継続的インテグレーション (CI).....	51
14. ビルドパイプラインとは	52
15. ビルドパイプラインの作成.....	53
16. サンプルコードの clone と Git の設定.....	62
17. ビルドと自動テストの実行.....	67
18. ビルドエラー時の挙動.....	80
19. テスト失敗とカンバンボードに Bug を追加	87
STEP 4. リリース管理.....	99
20. リリース定義とは	100
21. リリース定義の作成.....	101
22. ステージングへのデプロイとロードテスト.....	116
23. リリース環境へスワッピング	125
参考文献.....	132
24. Visual Studio Team Services に関する情報の入手元.....	133
25. その他の参考文献	134

1. はじめに



本自習書をご利用いただきありがとうございます。この自習書では、Microsoft が提供するパブリッククラウドサービスである **Visual Studio Team Services** を利用し、**Git** リポジトリと連動してアプリのビルドとテストが自動実行される “継続的インテグレーション” 環境を構築する一連の流れを、ハンズオン形式で学習体験します。

本自習書において、以下のサービスとソフトウェアを使用します。

◆ サービス

-  Visual Studio Team Services
-  Microsoft Azure
-  GitHub

◆ ソフトウェア

-  Visual Studio Code
-  Git

ソフトウェアに関しては、作業 PC の OS として **Windows 10** を想定しております。**macOS** で代用することも可能ですが、**macOS** での操作手順などは記載しておりませんので、操作が **Windows 10** と異なる場合は、都度置き換えて操作をお願いいたします。

STEP 1. 環境整備

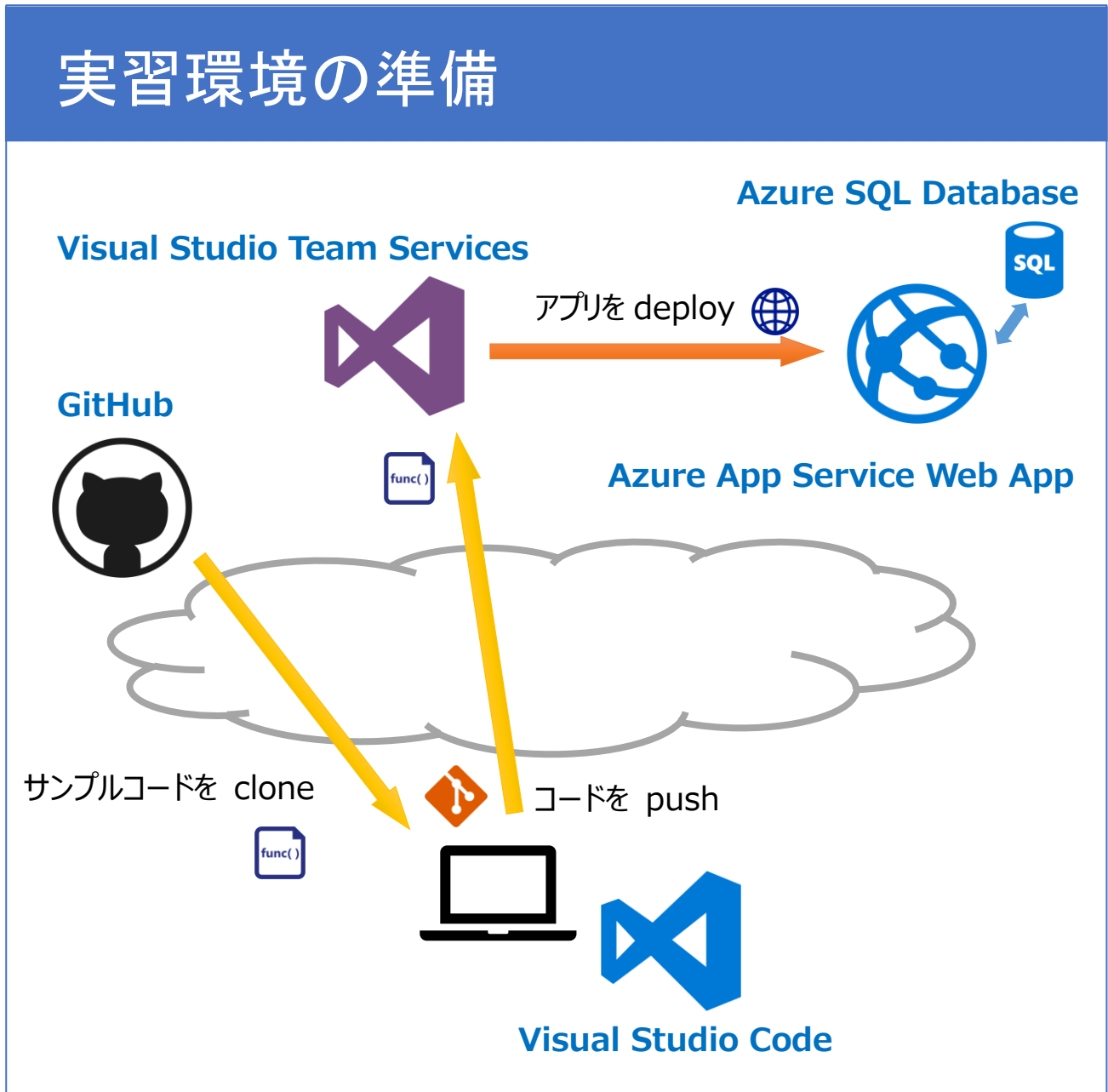
この STEP では、本自習書を試す環境準備について説明します。

この STEP では、次のことを学習します。

- ✓ Microsoft アカウントと Azure サブスクリプションの取得
- ✓ Visual Studio Team Services へのサインアップ
- ✓ 開発環境 (Visual Studio Code, Git) の構築
- ✓ Azure App Service Web App の作成
- ✓ Azure SQL Database の作成

2. 実習環境の準備

この自習書の手順は、次の実習環境を準備することで、実際に試すことができ、理解を深めることができます。



● GitHub

本自習書で使用するサンプルアプリ (Web アプリ) のリポジトリがあります。ここから、Visual Studio Code と Git を使用してサンプルアプリのコードを clone します。clone した後は、GitHub は使用しません。

●Visual Studio Team Services

Visual Studio Team Services は、チームによる開発をサポート、管理するクラウドサービスです。実習の中心となるサービスであり、Git リポジトリでのコード管理や、継続的インテグレーション (CI) を実現します。

●Azure App Service Web App

サンプルアプリを動かすための PaaS です。Microsoft Azure のサービスのひとつであり、Microsoft Azure にサインインするには、Microsoft アカウントが必要になります。

●Azure SQL Database

サンプルアプリのデータを管理するためのクラウドデータベースです。同じく、Microsoft Azure のサービスのひとつです。

●Visual Studio Code

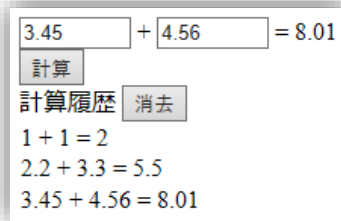
Windows 10 や macOS などのデスクトップで、コードを編集するためのコードエディターです。このエディターを使用してサンプルアプリのコードを変更し、Visual Studio Team Services の Git リポジトリに push します。

3. サンプルアプリについて

サンプルアプリは、Visual Studio 2017 にて、C# 言語で実装された ASP.NET Web アプリです。以下の Github リポジトリにあり、誰でも clone できるようになっています。

<https://github.com/creationline/vsts-handson-webapp>

アプリの動作としては、浮動小数点数の足し算を計算し、結果を表示します。また、計算の履歴をデータベースで管理し、ページ上にリスト表示します。



計算ロジックは、文字列を数値に変換し、足し算して返す仕組みになっています。

(HandsOnWebApp/Default.aspx.cs)

```
/// <summary>
/// 文字列の数値から、足し算の結果を返します。
/// </summary>
/// <param name="s1">足される数を表す文字列。</param>
/// <param name="s2">足す数を表す文字列。</param>
/// <returns>足し算の結果。</returns>
private float Calculate(string s1, string s2) {
    float f1 = float.Parse(s1);
    float f2 = float.Parse(s2);
    return f1 + f2;
}
```

また、計算ロジックの Unit Test が記述されています。

(HandsOnWebApp.Tests/UnitTest.cs)

```
[TestClass]
public class UnitTest {

    [TestMethod]
    public void TestMethod1() {
        var pvObj = new PrivateObject(new Default());
        var result = pvObj.Invoke("Calculate", "1", "2");
        Assert.AreEqual(3f, result);
    }

    [TestMethod]
    public void TestMethod2() {
        var pvObj = new PrivateObject(new Default());
        try {
            pvObj.Invoke("Calculate", "2", "2");
        } catch (FormatException) {
            return;
        }
        Assert.Fail();
    }
}
```

本自習書では、このサンプルアプリのコードを使用し、Visual Studio Team Services にてコード管理、ビルド、テストを行います。

4. Microsoft Azure サブスクリプションの準備

この自習書を進めるには、Microsoft Azure サブスクリプションをあらかじめ契約しておく必要があります。

既に有効な Microsoft アカウント および Microsoft Azure サブスクリプションをお持ちの場合、この事前作業はスキップしてください。

※サブスクリプションの管理者権限がある **Microsoft アカウント**が必要になります。

ワンポイント

Microsoft Azure サブスクリプション作成時に、確認コードを音声または SMS で受け取るための携帯電話、および身元確認のためのクレジットカードが必要になります。

●Microsoft アカウントの準備

URL <http://www.microsoft.com/ja-jp/msaccount/signup/default.aspx> を Web ブラウザーで開き、新しく Microsoft アカウントを作成します。

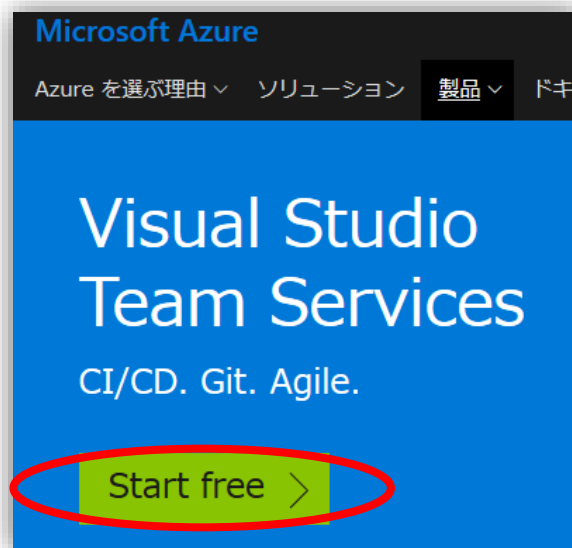
●Microsoft Azure サブスクリプションの作成

<http://msdn.microsoft.com/ja-jp/windowsazure/ee943806.aspx> を Web ブラウザーで開き、手順に従って Microsoft Azure サブスクリプションを作成します。

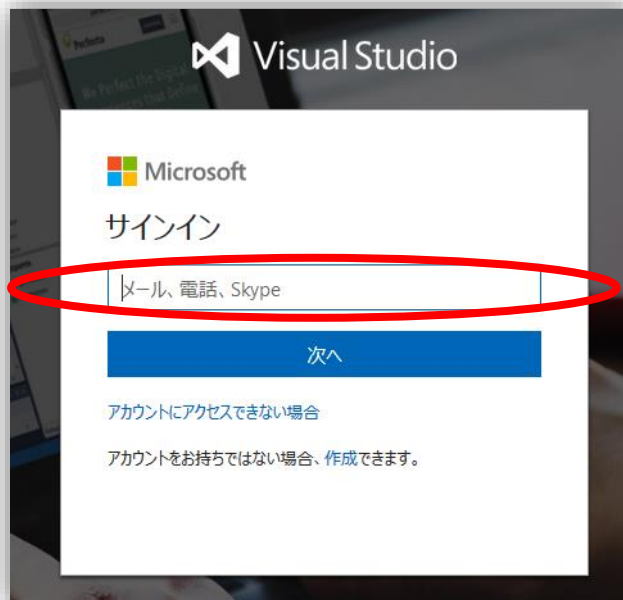
5. Visual Studio Team Services へのサインアップ

次の手順では、Microsoft アカウントを使用して Visual Studio Team Services にサインアップします。

1. Web ブラウザーを起動して Visual Studio Team Services のトップページ <https://azure.microsoft.com/ja-jp/services/visual-studio-team-services/>を開き、[Start free] をクリックします。



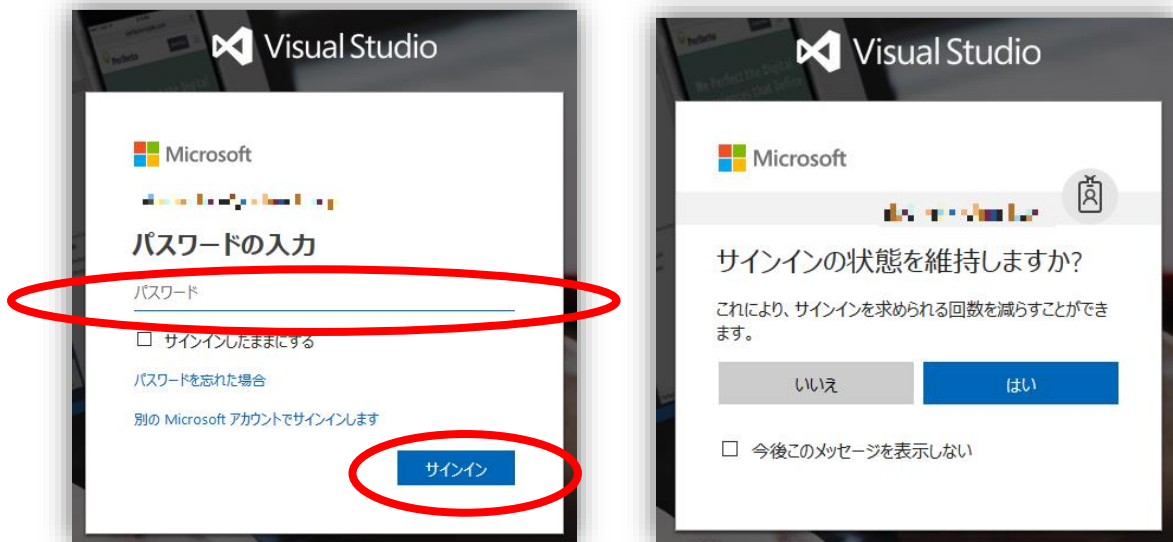
2. Microsoft アカウントのメールアドレスを入力し、Tab キーを押下します。



3. 職場用の Microsoft アカウントの場合は、引き続き、パスワードを入力して [サインイン] をクリックします。

個人用 Microsoft アカウントの場合は以下のページへリダイレクトするので、ここでパスワードを入力し、[サインイン] をクリックします。

※サインイン後、サインインの状態を維持するかを問われる場合がありますが、任意で [はい] か [いいえ] のどちらかを選択します。



4. Visual Studio Team Services にアクセスするためのドメイン (の第 3 レベル) を入力します。ここでは、**vsts-handson-0001** とします。

※数字の部分は被らないように、適当な値に変更してください。

The image shows a screenshot of the 'プロジェクトをホストする地域:' (Project hosting region) screen in Visual Studio. The input field for the domain 'vsts-handson-0001' is circled in red. Below the input field, there are radio buttons for 'Git' (selected) and 'Team Foundation バージョン管理' (unselected). At the bottom, there is a '続行' (Next) button.

5. [詳細の変更] をクリックします。ここでは、サインインと同時に作成される最初のプロジェクトのプロジェクト名などを入力します。パラメーターは以下のように入力します。

コードの管理に次を使用:

☒ Git

☐ Team Foundation バージョン管理

プロジェクト名:

HandsonProject

以下を使用して作業を整理します:

Scrum

プロジェクトをホストする地域:

米国中央部

パラメーター	説明	今回の設定
コード管理に次を使用	SCM の種類を選択します。	Git
プロジェクト名	プロジェクトの名前です。	HandsonProject
以下を使用して作業を整理します	開発手法を選択します。	Scrum
プロジェクトをホストする地域	プロジェクトを置く地域です。	米国中央部

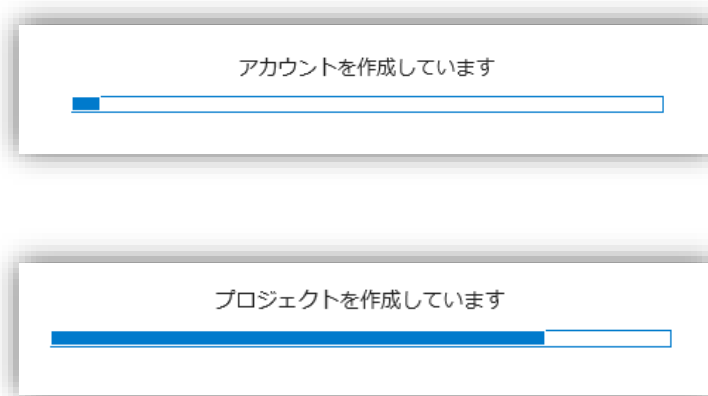
6. [続行] をクリックします。

プロジェクトをホストする地域:

米国中央部

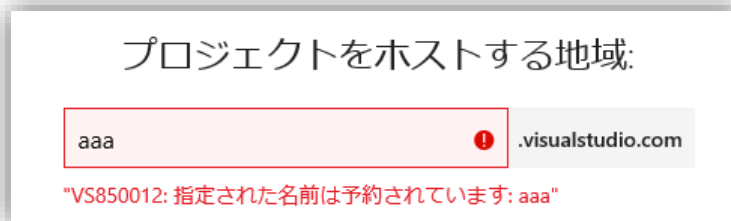
続行

7. Visual Studio Team Services のアカウントと、最初のプロジェクトの作成が開始されます。

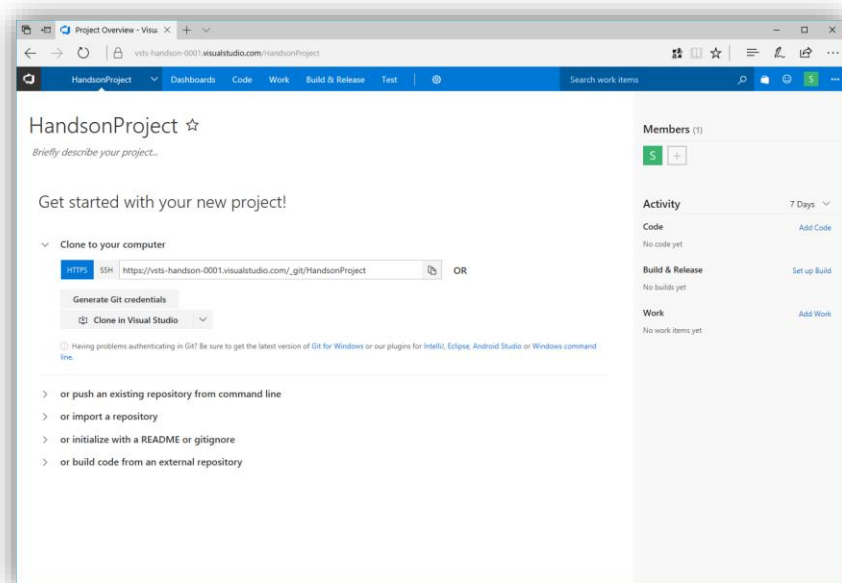


ワンポイント

ドメイン名が他と被った場合は、以下のようなエラーが表示されます。この場合は、**vsts-handson-0001** の数字の部分の適当な値に変更後、「続行」をクリックしてください。



8. 以下の画面が表示されれば、サインアップは完了です。

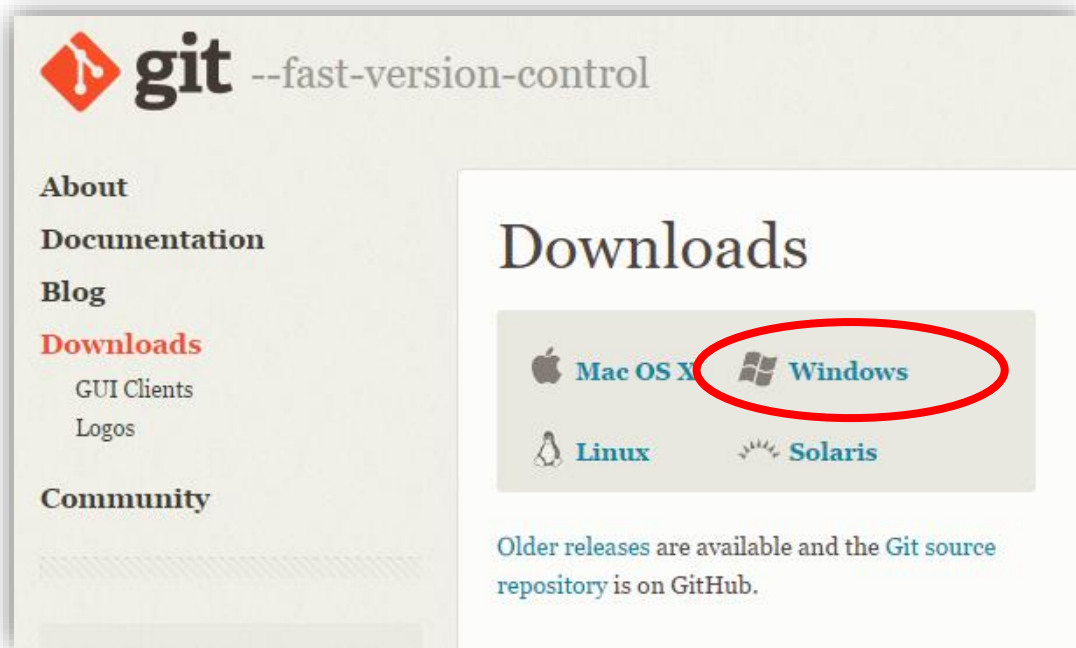


6. Git のインストール

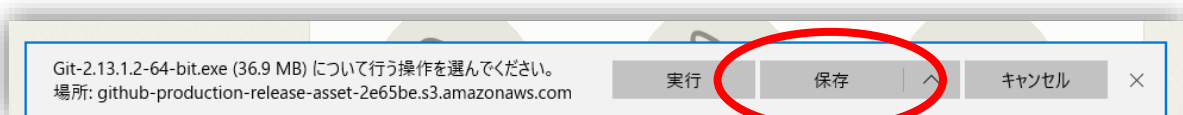
Source Code Management として Git を使用します。次の手順では、Windows コンピューターに Git をインストールします。

※macOS では標準でインストール済みのため、この手順は不要です。

1. Git のダウンロードページ <https://git-scm.com/downloads> を開き、[Windows] をクリックします。



2. ダウンロードの操作が求められるので、[保存] をクリックします。

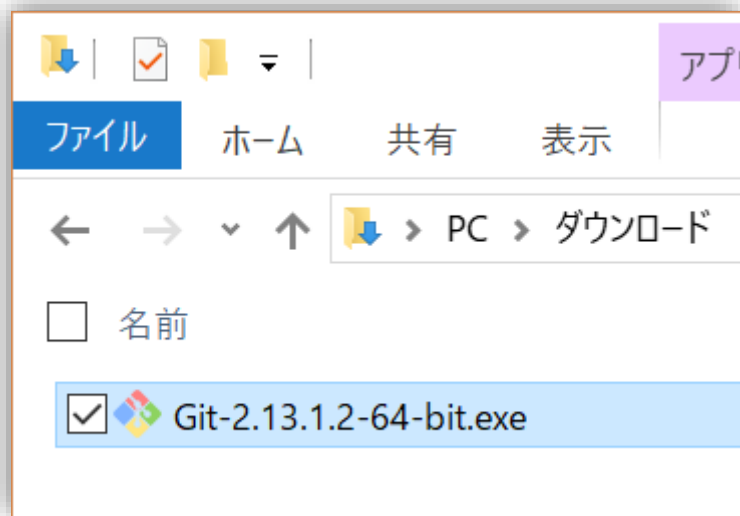


ワンポイント

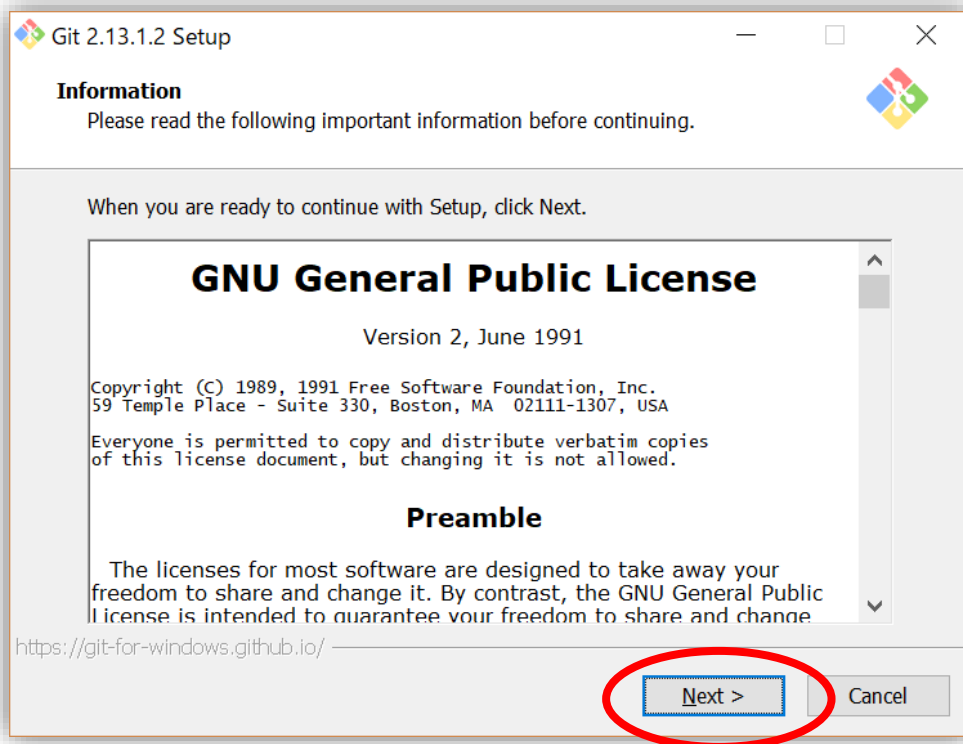
上図は、Edge でダウンロードしようとしているところです。使用する Web ブラウザーでダウンロードの操作が異なります。

また、Git のバージョンは随時アップデートされるので、実際にダウンロードされるバージョンは上図とは異なる可能性があります。

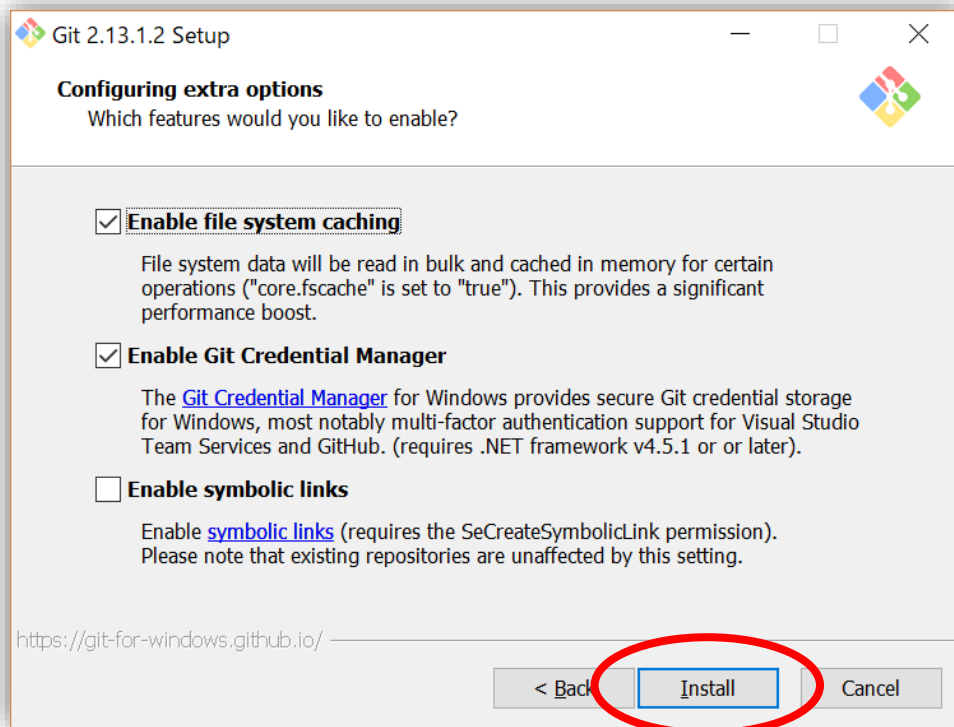
- ダウンロードしたファイルを実行します。



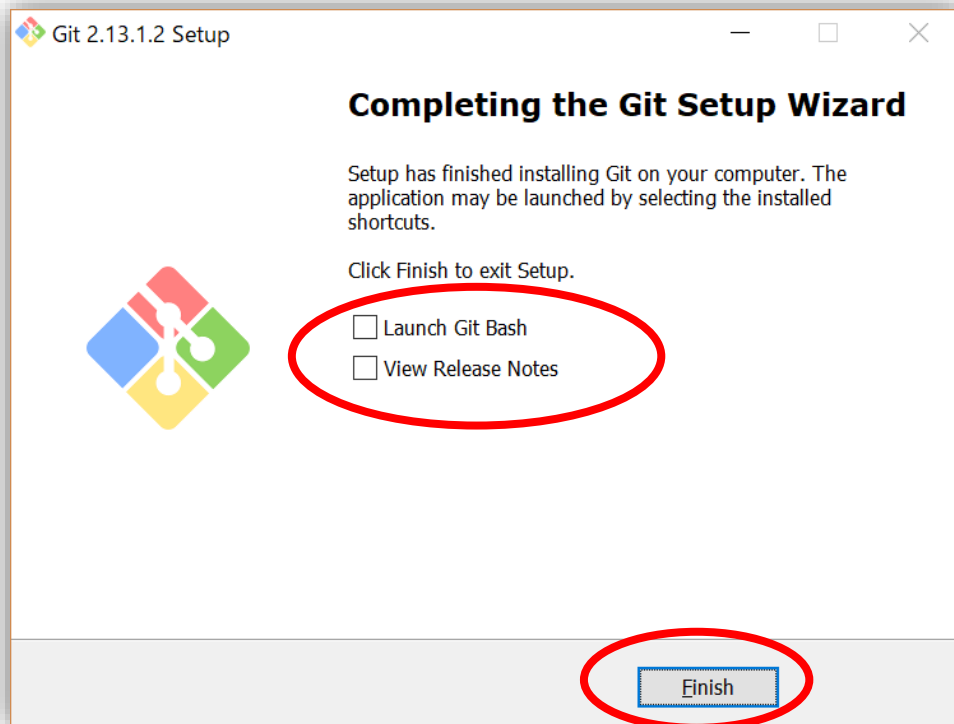
- セットアップウィザードが起動するので、特に設定は変更せず [Next] をクリックして画面を進めていきます。



5. セットアップウィザードの最後の画面で [Install] をクリックします。

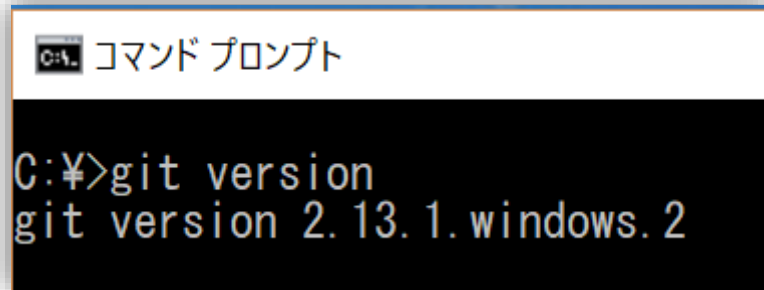


6. インストール完了後、[Launch Git Bash] [View Release Notes] のチェックをはずし、[Finish] をクリックします。



7. コマンドプロンプトを起動し、以下のコマンドを入力します。**git** コマンドが動作することを確認します。

```
git version
```



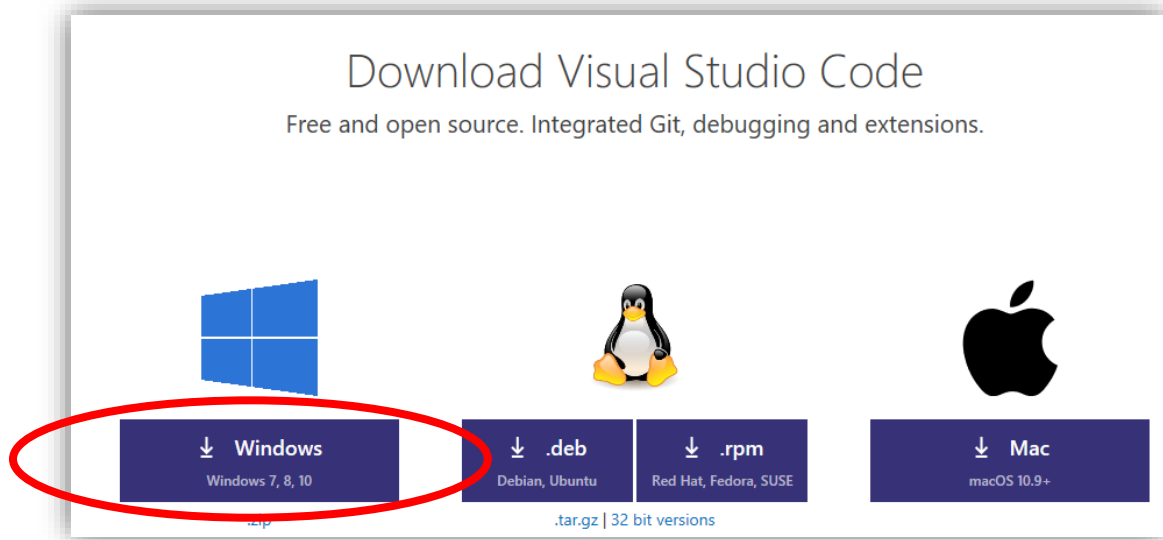
これで、Git のインストールは完了です。

7. Visual Studio Code のインストール

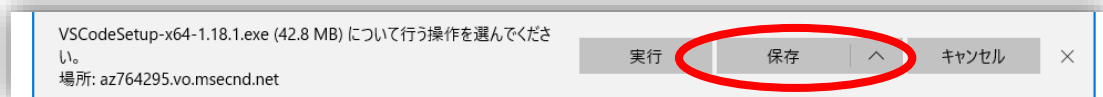
Visual Studio Code は、サンプルアプリのコード編集に使用します。次の手順では、Windows コンピューターに Visual Studio Code をインストールします。

※macOS では、インストール手順が異なります。<https://code.visualstudio.com/docs/setup/mac> を参考にインストールしてください。

1. Visual Studio Code のダウンロードページ <https://code.visualstudio.com/download> を開き、[Windows] をクリックします。



2. ダウンロードの操作が求められるので、[保存] をクリックします。

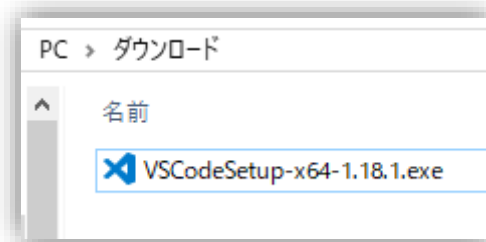


ワンポイント

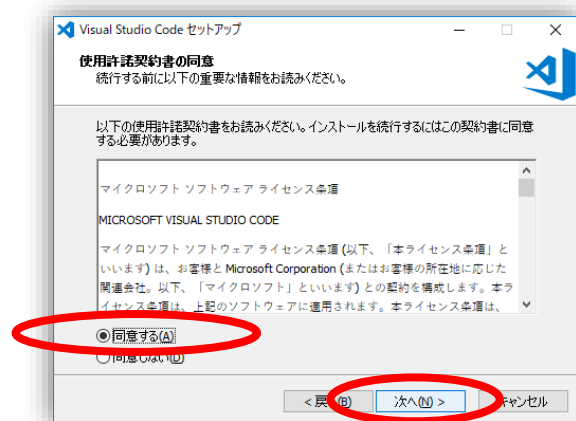
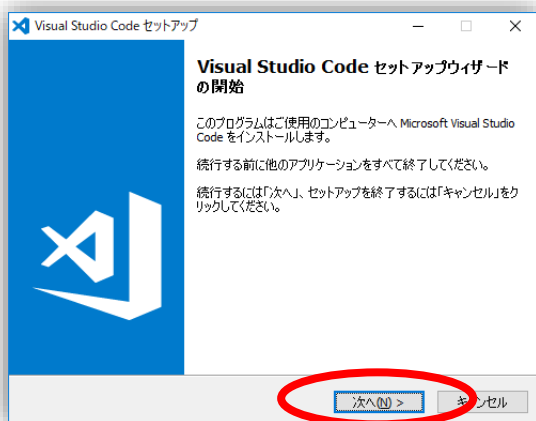
上図は、Edge でダウンロードしようとしているところです。使用する Web ブラウザーでダウンロードの操作が異なります。

また、Visual Studio Code のバージョンは随時アップデートされるので、実際にダウンロードされるバージョンは上図とは異なる可能性があります。

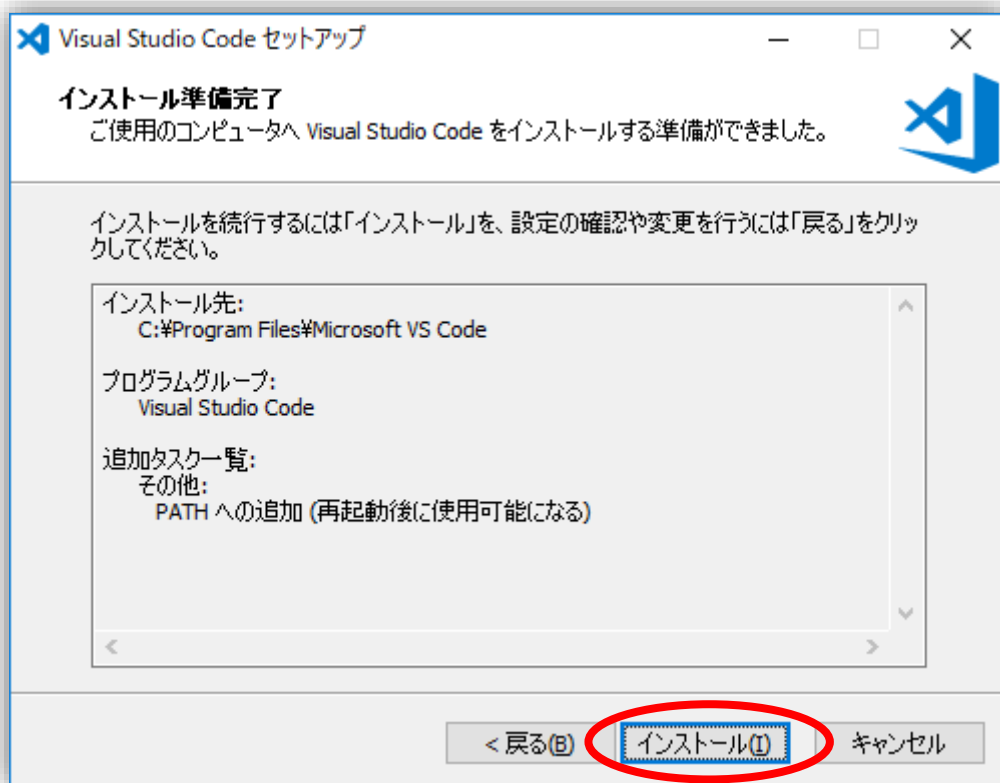
3. ダウンロードしたファイルを実行します。



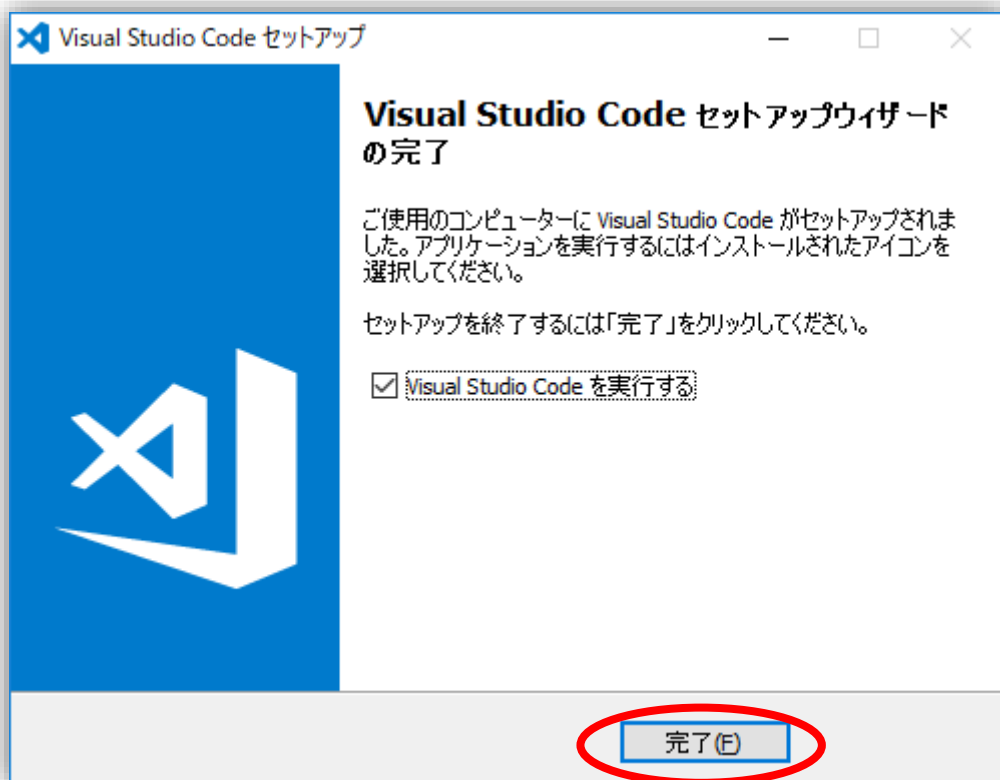
4. セットアップウィザードが起動します。[使用許諾契約書の同意] の画面で [同意する] を選択し、以降は [次へ] をクリックして画面を進めていきます。



5. セットアップウィザードの最後の画面で [インストール] をクリックします。



6. インストール完了後、[完了] をクリックします。



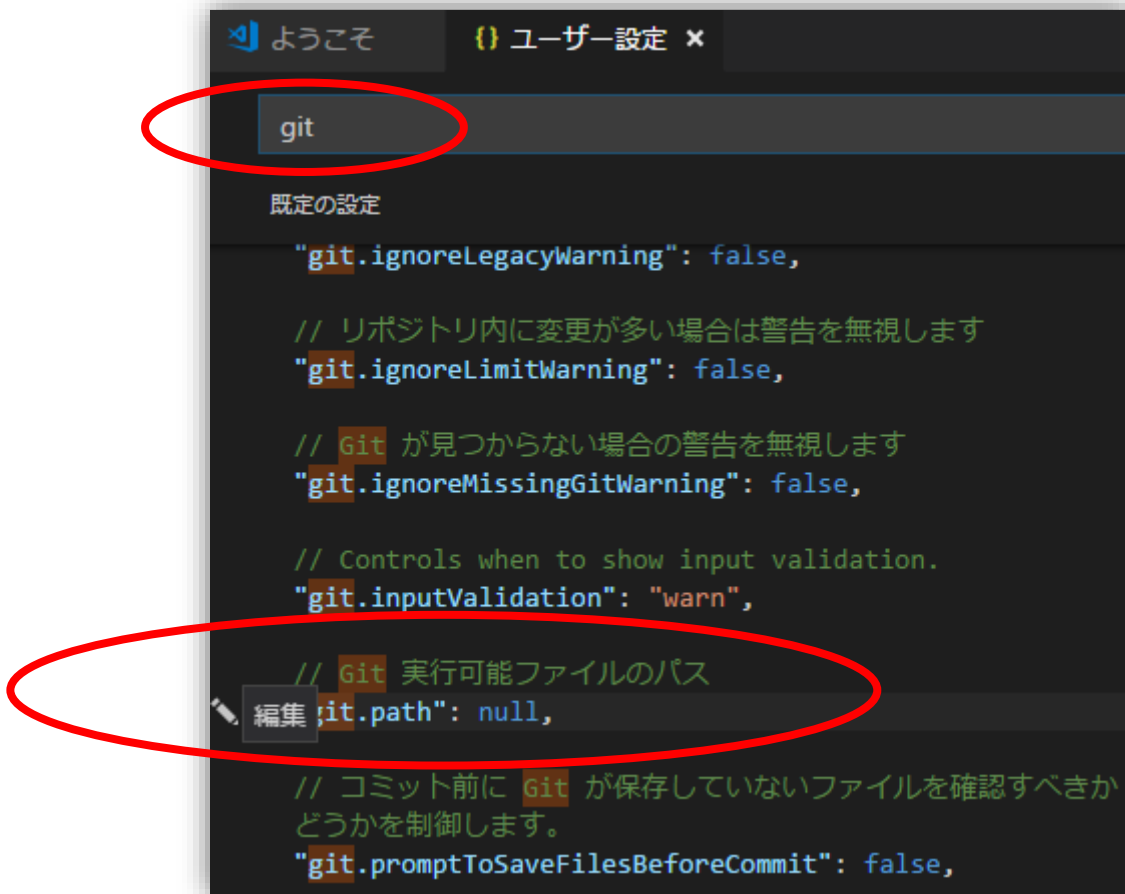
7. Visual Studio Code が開くことを確認します。



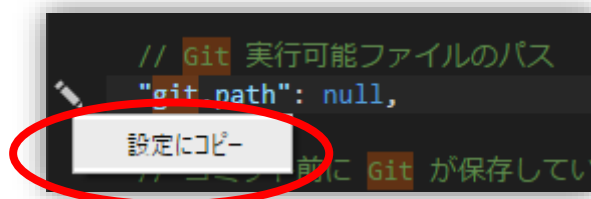
8. Visual Studio Code から Git の操作が行えるように、git コマンドのパスを設定ファイルに登録します。メニューバーから、[ファイル] – [基本設定] – [設定] をクリックします。



9. ユーザー設定が開くので、検索ボックスに「git」と入力し、設定項目[git.path] の [編集] をクリックします。

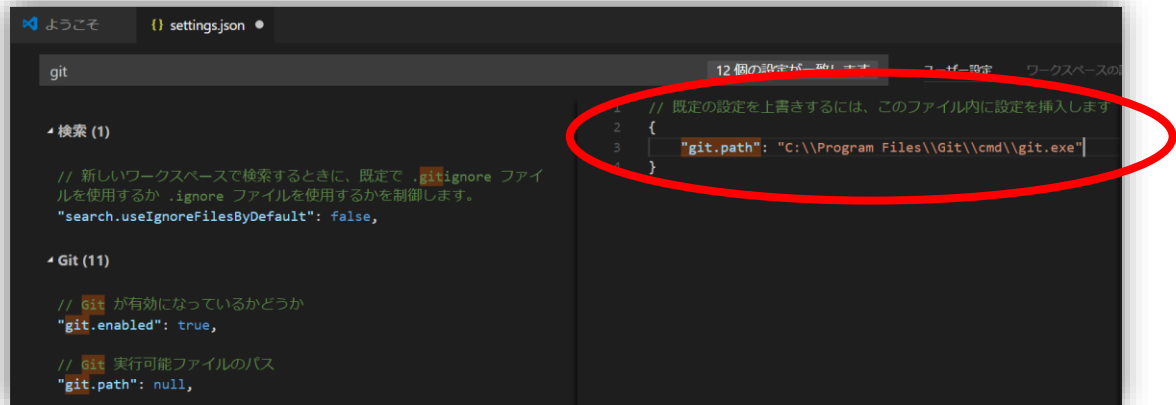


10. [設定にコピー] ボタンが表示されるので、クリックします。



11. 画面右側に設定項目 [git.path] が表示されるので、git コマンドのパスを入力します。

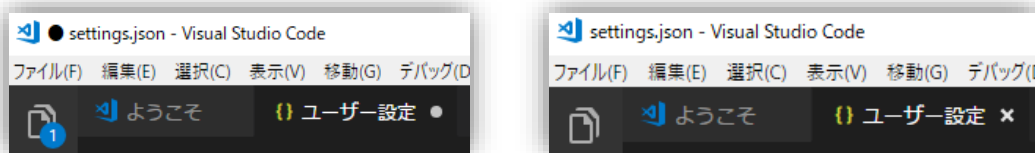
git コマンドのパス: "C:¥¥Program Files¥¥Git¥¥cmd¥¥git.exe"



ワンポイント

パスの区切り文字 ¥ は、2 つ重ねて入力する必要があります。また、このパスは Windows の場合であり、macOS の場合は正しいパスに置き換えてください。

12. Ctrl + S キーを押下して、ファイルを保存します。



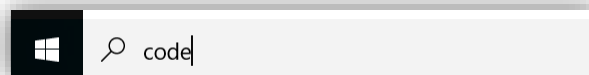
ワンポイント

保存されると、ファイル名の横にある ● が消えます。

13. Visual Studio Code を一度終了し、起動しなおします。再起動後に、git コマンドのパス設定が有効になります。

ワンポイント

Windows 10 の場合は、Cortana から 「code」 と入力すると簡単に起動できます。



これで、Visual Studio Code のインストールは完了です。

8. Azure App Service Web App の作成

Azure App Service Web App は、サンプルアプリをデプロイするために使用します。次の手順では、Microsoft Azure にサインインして、新しい Web App を作成します。

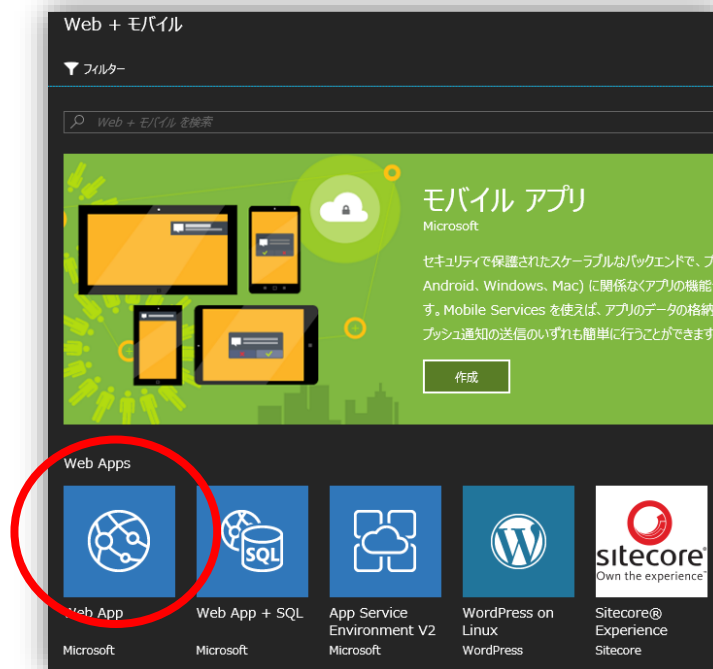
1. Azure ポータル <https://portal.azure.com/> に Microsoft アカウントでサインインします。



2. 左メニューから [App Service] をクリックし、App Service のメニューから [追加] をクリックします。



3. [Web + モバイル] ブレードの [Web App] をクリックします。




4. [Web App] ブレードの [作成] をクリックします。



5. 各パラメーターを以下のように入力し、[作成] をクリックします。

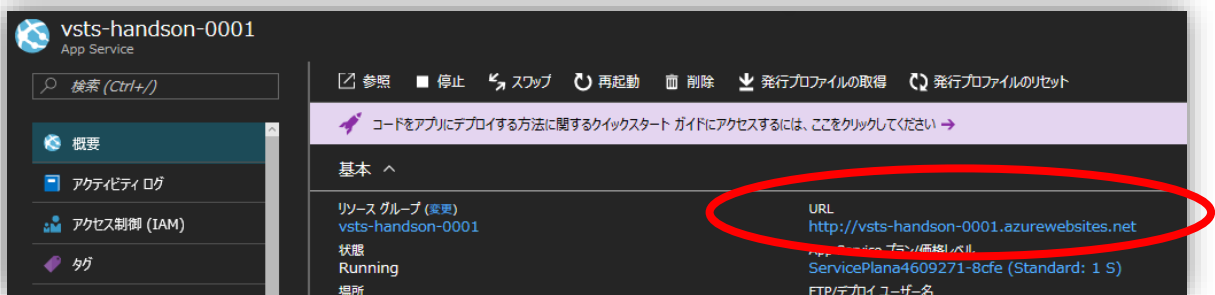
パラメーター	説明	今回の設定
アプリ名	アプリの名前です。	vsts-handson-0001 ※
サブスクリプション	使用するサブスクリプションです。	<任意>
リソースグループ	Web App を格納するリソースグループです。	[新規作成] を選択し、vsts-handson-0001 ※
OS	Web App の OS です。	Windows
App Service プラン/場所	Web App のサービスプランと作成される場所です。	<規定値>
Application Insights	アプリのパフォーマンス管理と分析を行うサービスです。	オフ

※アプリ名とリソースグループ名は、わかりやすくするために、Visual Studio Team Services のドメインで指定した値と同じにします。例えば、**Visual Studio Team Services** のドメインが **vsts-handson-0002** の場合は、アプリ名とリソースグループ名も **vsts-handson-0002** にします。

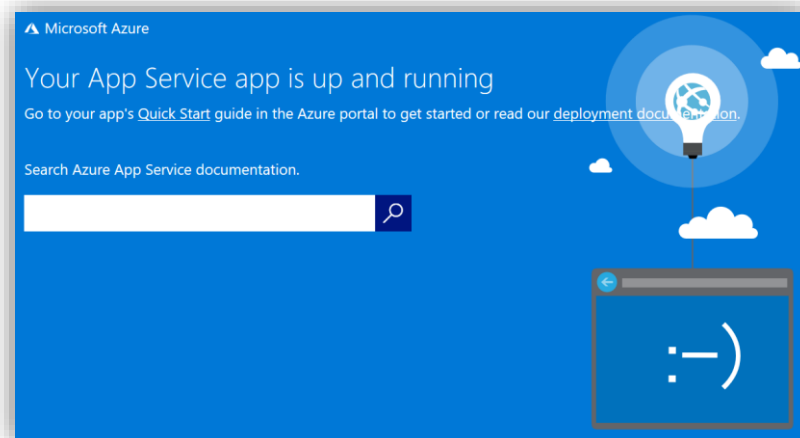
6. 作成完了後、ツールバーの  をクリックし、[展開が成功しました] の通知の [リソースに移動] をクリックします。



7. Web App が作成されていることを確認します。また、URL をクリックして、Web ページが表示されることを確認します。



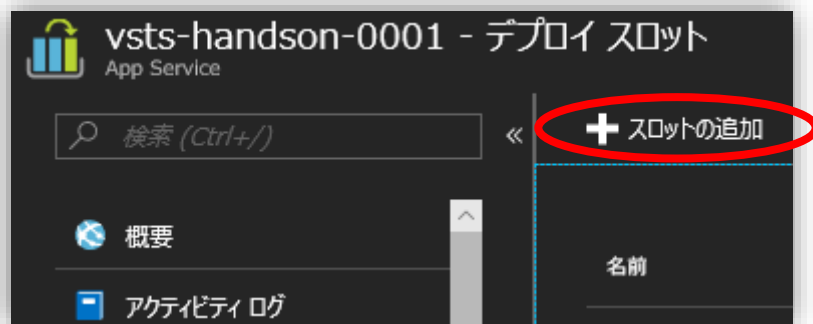
8. デフォルトの Web ページが表示されます。



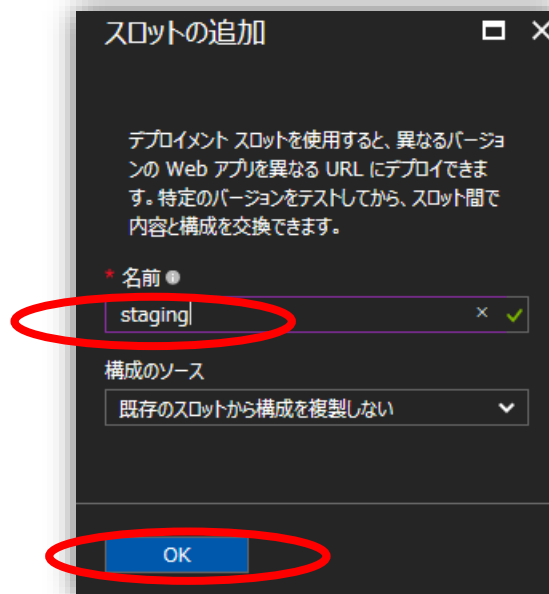
9. Web App のメニューから、[デプロイ スロット] をクリックします。



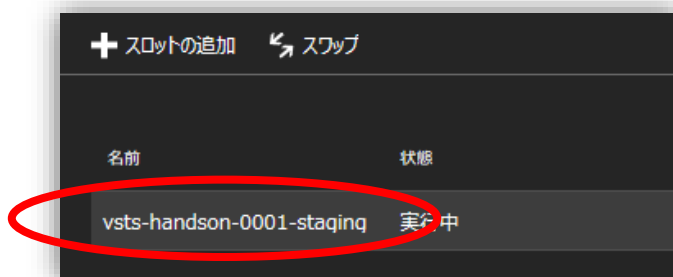
10. [スロットの追加] をクリックします。



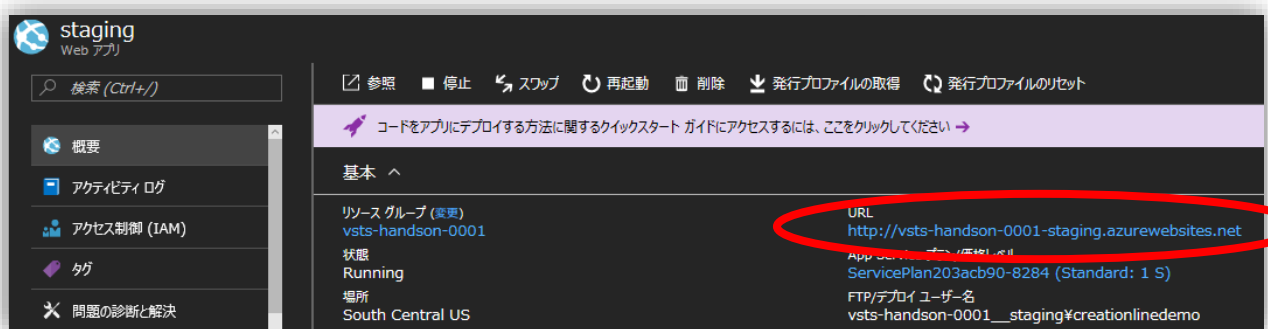
11. [スロットの追加] ブレードが表示されるので、[名前] に「staging」と入力し、[OK] をクリックします。



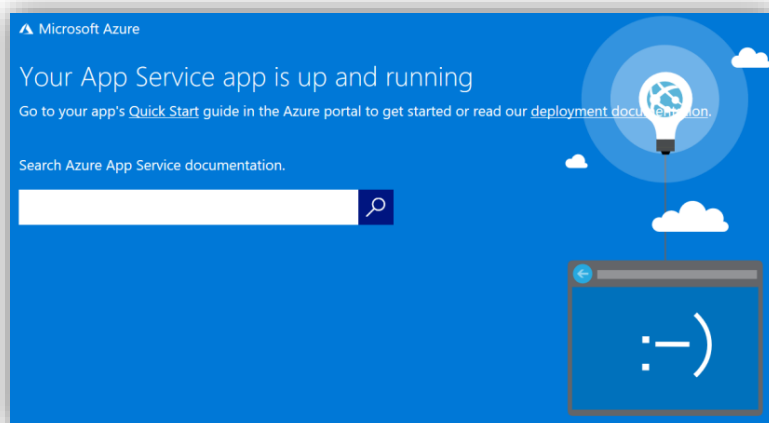
12. ステージング用のスロットが追加されるので、そのスロットをクリックします。



13. ステージングの概要が表示されるので、URL をクリックして Web ページが表示されることを確認します。



14. デフォルトの Web ページが表示されます。これで、既定の運用スロットとステージングスロットの2つの環境が揃いました。



ワンポイント

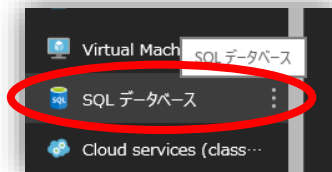
ステージングスロットは、既定のスロット（本番運用環境）と同等のスペックを持ち、主に運用に入る前のテスト（機能テスト、ロードテストなど）で使われます。

これで、Azure App Service Web App の作成は完了です。

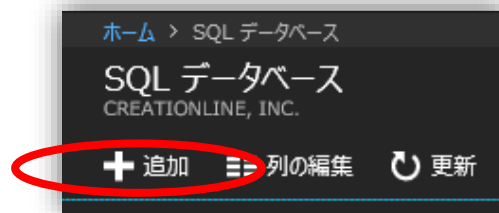
9. Azure SQL Database の作成

Azure SQL Database は、サンプルアプリのデータを管理するために使用します。次の手順では、Azure ポータルから SQL Database を作成します。データベースは、運用用とステージング用の 2 つを作成します。

1. 引き続き、Azure ポータルの左メニューから、[SQL データベース] をクリックします。



2. まずは、運用用のデータベースから作成します。SQL データベースのメニューから [追加] をクリックします。



3. SQL Database を作成するためのブレードが表示されます。まずは、[サブスクリプション] に Web App 作成時に指定したものと同じサブスクリプションを指定します。
続いて、データベースを動かす SQL サーバーのインスタンスから作成します。[サーバー 必要な設定の構成] をクリックします。



4. 新しいサーバーを作成するためのブレードが表示されるので、各パラメーターを以下のように入力し、[選択] をクリックします。

[サーバー管理者ログイン] と [パスワード] は、後程 **Web App** にデータベースの接続文字列を追加する際に必要になるので、忘れないように控えておいてください。

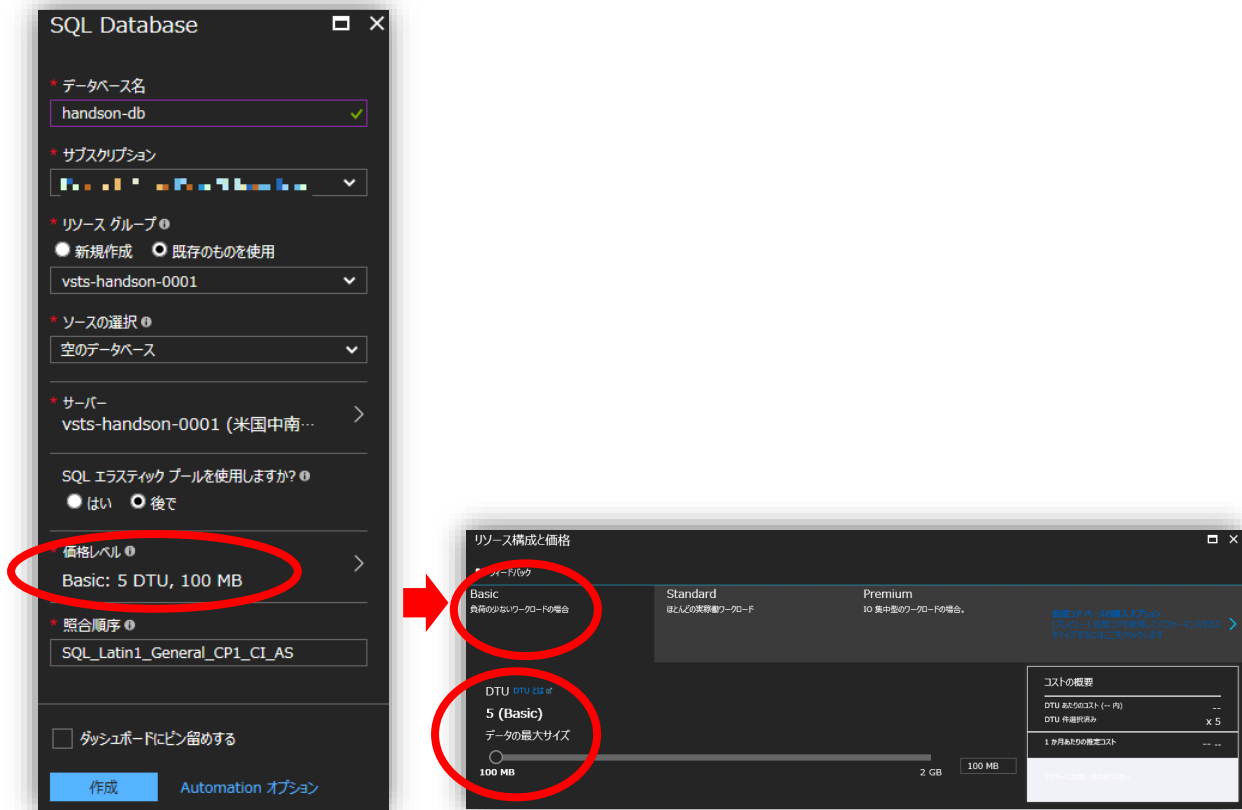
パラメーター	説明	今回の設定
サーバー名	サーバーの名前です。	vsts-handson-0001 ※
サーバー管理者ログイン	サーバーの管理者名です。	<任意>
パスワード	サーバー管理者のパスワードです。	<任意>
パスワードの確認	パスワードの確認です。	パスワードと同じ値
場所	サーバーが作成される場所です。	<規定値>
Azure サービスにサーバーへのアクセスを許可する	Web App などのサービスが、このサーバーにアクセスできるようにします。	チェックする

※サーバー名は、わかりやすくするために、Visual Studio Team Services のドメインで指定した値と同じにします。例えば、**Visual Studio Team Services** のドメインが **vsts-handson-0002** の場合は、サーバー名も **vsts-handson-0002** にします。

5. 引き続き、データベースの各パラメーターを以下のように入力します。

[価格レベル] は、これをクリックするとリソース構成と価格を設定するブレードが表示されます。ここで [Basic] を選択、[データの最大サイズ] のスライダーを「100 MB」にし、[適用] をクリックします。

すべての設定が終わったら、[作成] をクリックします。



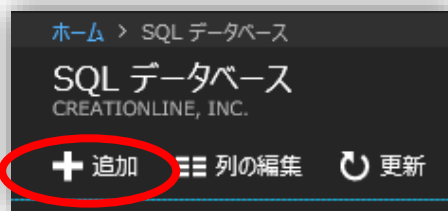
パラメーター	説明	今回の設定
データベース名	データベースの名前です。	handson-db
サブスクリプション	使用するサブスクリプションです。	手順 3 で指定済み
リソースグループ	データベースのリソースグループです。	[既存のものを使用] を選択し、vsts-handson-0001 ※
ソースの選択	サンプルデータベースを使用するかを選択します。	空のデータベース
サーバー	データベースを作成するサーバーです。	手順 4 で作成したサーバー
SQL エラスティックプールを使用しますか?	エラスティックプールを使用するかどうかを選択します。	後で
価格レベル	データベースのパフォーマンスを設定します。	Basic: 5DTU, 100 MB
照合順序	データソートの規則を設定します。	<規定値>

※リソースグループは、Web App と同じリソースグループを使用します。例えば、Web App で作成したリソースグループが vsts-handson-0002 の場合は、データベースでも vsts-handson-0002 のリソースグループを使用します。

6. データベースの作成が開始されます。作成には時間がかかるので、しばらく待ちます。



7. 作成完了後、ステージング用のデータベースを作成します。[追加] をクリックします。



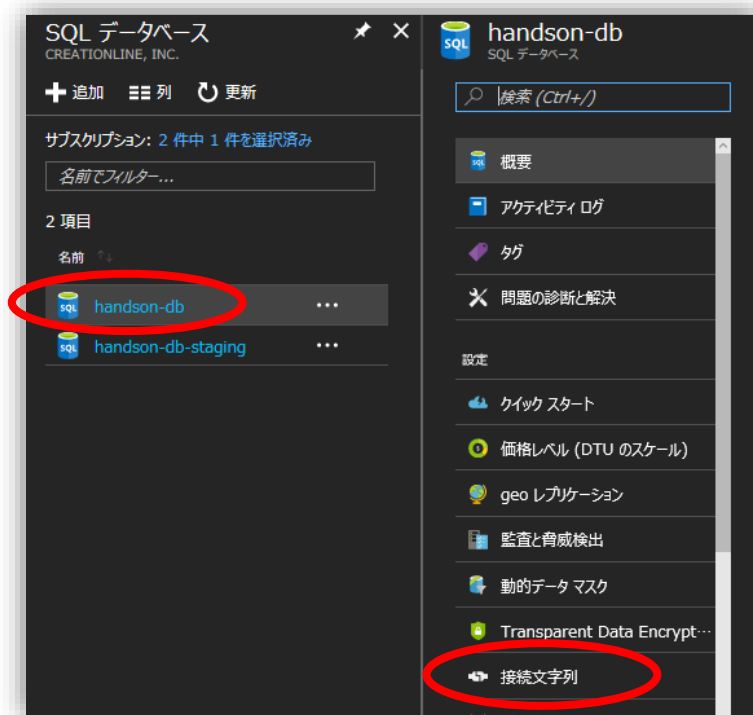
8. [データベース名] に「handson-db-staging」と入力します。他のパラメーターは手順 5 と同じにします。設定後、[作成] をクリックします。



9. 作成完了後、[更新] をクリックします。作成したデータベースの一覧が表示されます。

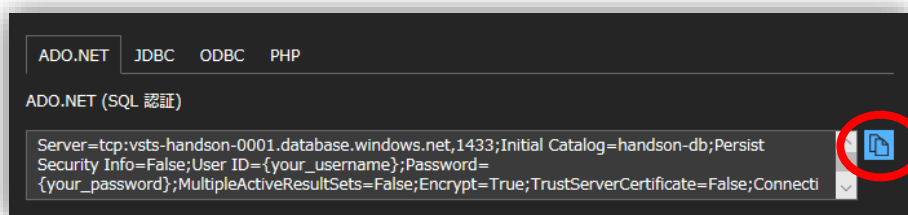


10. Web App からこのデータベースに接続するために、接続文字列を取得します。[handson-db] – [接続文字列] をクリックします。

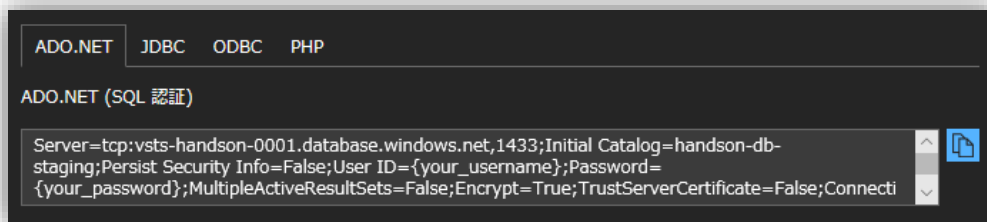
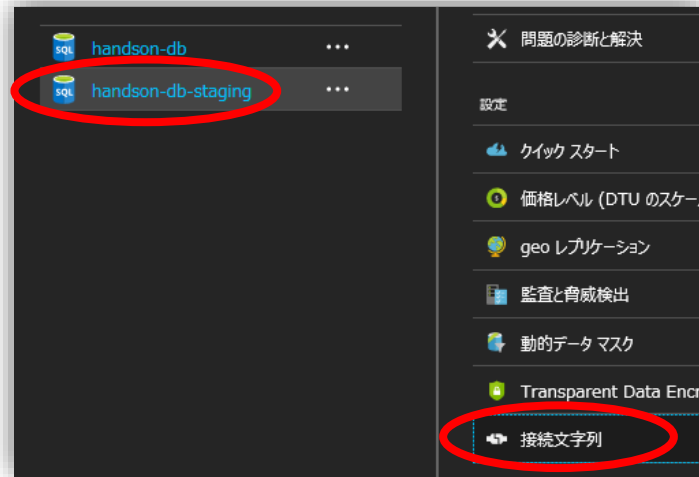


11. handson-db の接続文字列が表示されますので、ADO.NET (SQL 認証) の接続文字列を控えておきます。テキスト右側の青いアイコンをクリックするとクリップボードにコピーできますので、適当なテキストファイルに貼り付けます。

このとき、**{your_username}** と **{your_password}** の部分を、手順 4 で控えておいた【サーバー管理者ログイン】と【パスワード】の値に置き換えてください。



12. 同様に、[handson-db-staging] – [接続文字列] をクリックして、handson-db-staging の接続文字列を取得します。この ADO.NET (SQL 認証) の接続文字列もテキストファイルにコピーしておきます。このとき、**{your_username}** と **{your_password}** の部分を、手順 4 で控えておいた【サーバー管理者ログイン】と【パスワード】の値に置き換えてください。



ワンポイント

handson-db と handson-db-staging の接続文字列の違いは、Catalog の値だけです。

```
Server=tcp:vsts-handson-0001.database.windows.net,1433;Initial Catalog=handson-db;Persist Security Info=False;User ID={your_username};Password={your_password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;
```

13. これでデータベースの作成は完了しました。引き続き、接続文字列を Web App に追加する作業を行います。

左メニューから、[App Service] – [vsts-handson-0001] をクリックします。

※**vsts-handson-0001** は、Web App を作成したアプリ名に置き換えてください。



14. Web App のメニューから [アプリケーション設定] をクリックします。



15. [接続文字列] の [新しい接続文字列の追加] をクリックすると入力フィールドが表示されるので、パラメーターを以下のように入力します。[値を入力してください] には、先ほどテキストファイルにコピーした **handson-db** の接続文字列を貼り付けます。

パラメーター	説明	今回の設定
名前の入力	接続文字列の名前です。	handsondb
値を入力してください	接続文字列です。	手順 11 の handson-db の接続文字列
接続の種類	接続文字列の種類です。	SQLAzure
スロットの設定	スロット固有の設定値かどうかを表します。これを有効にすると、別のスロットとスワッピングを行っても、この設定値は固定されます。(交換されません)	チェックする

ワンポイント

ここで設定した接続文字列は、ASP.NET では Web.config ファイルの connectionStrings 要素に相当します。詳しくは以下を参照してください。

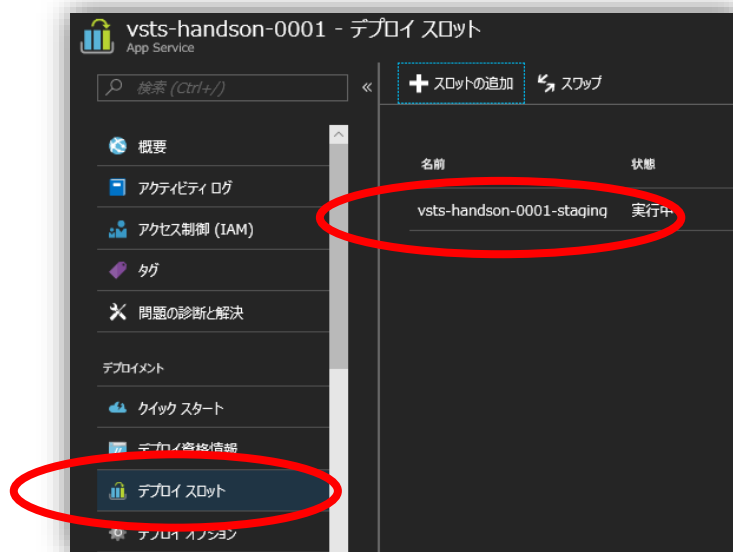
[https://msdn.microsoft.com/en-us/library/bf7sd233\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/bf7sd233(v=vs.85).aspx)

16. 入力後、[保存] をクリックします。



17. 続けて、Web App のステージング用のスロットに接続文字列を追加します。メニューから [デプロイ スロット] – [vsts-handson-0001-staging] をクリックします。

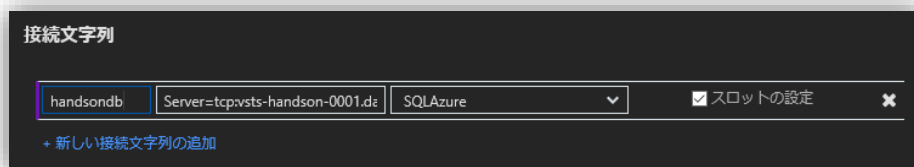
※**vsts-handson-0001** は、Web App を作成したアプリ名に置き換えてください。



18. staging スロット用のメニューが表示されるので、[アプリケーション設定] をクリックします。

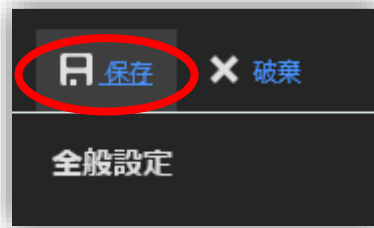


19. [接続文字列] の [新しい接続文字列の追加] をクリックすると入力フィールドが表示されるので、パラメーターを以下のように入力します。[値を入力してください] には、先ほどテキストファイルにコピーした **handson-db-staging** の接続文字列を貼り付けます。



パラメーター	今回の設定
名前の入力	handsondb
値を入力してください	手順 12 の handson-db-staging の接続文字列
接続の種類	SQLAzure
スロットの設定	チェックする

20. 入力後、[保存] をクリックします。



これで Web App への接続文字列の追加は完了です。アプリとデータベースの関連付けは以下のようになっています。

Web App	SQL Database
vsts-handson-0001	handson-db
vsts-handson-0001-staging (vsts-handson-0001 の追加スロット)	handson-db-staging

※**vsts-handson-0001** は、Web App を作成したアプリ名に置き換えてください。

STEP 2. タスク管理

この STEP では、Visual Studio Team Services のカンバンボード（バックログ）を使用してのタスク管理について説明します。

この STEP では、次のことを学習します。

- ✓ カンバンボードでのタスク管理

10. カンバンボードでのタスク管理

Visual Studio Team Services のカンバンボード (バックログ) は、Agile、CMMI、Scrum の開発手法に対応しており、チーム内でプロジェクトのタスク管理ができます。

開発手法が Scrum の場合、基本的に以下の関係で Work item が構成されています。



● Feature

この Work item には、実装する機能や達成したい事象などを記述します。

● Backlog Item

Feature 配下に置かれる Work item であり、Scrum では Product Backlog (Story) を表します。Feature を実現するための要素を記述します。Product Backlog には優先順位があり、上位のものほど優先順位が高くなります。

また、テストなどで問題点があった場合は、Bug を作成できます。

● Task

Backlog Item 配下に置かれる Work item であり、担当者が実際に行う作業を記述します。

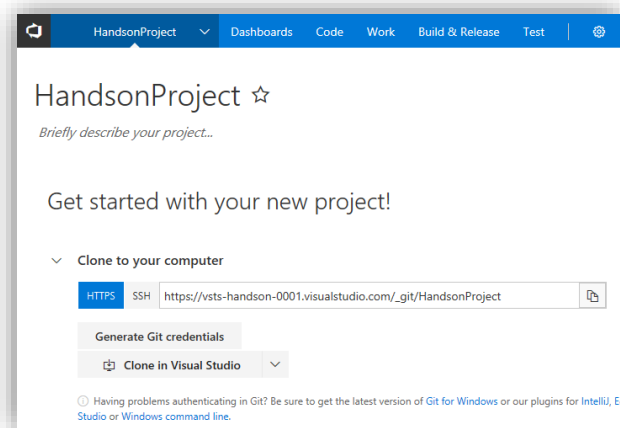
本自習書では、実習で行う作業をタスクとしてカンバンボードで管理します。開発手法は Scrum を使用しますが、目的は簡易的なタスク管理の学習であり、厳密な Scrum 開発は行いません。

11. Feature と Backlog Item の追加

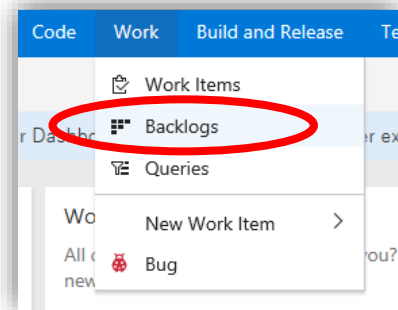
次の手順では、カンバンボードに Feature と Backlog Item を追加します。

1. Web ブラウザーで、Visual Studio Team Services の HandsonProject を開きます。
※ドメイン が **vsts-handson-0001** の場合、以下の URL になります。Visual Studio Team Services サインアップ時に指定したドメインに置き換えてください。

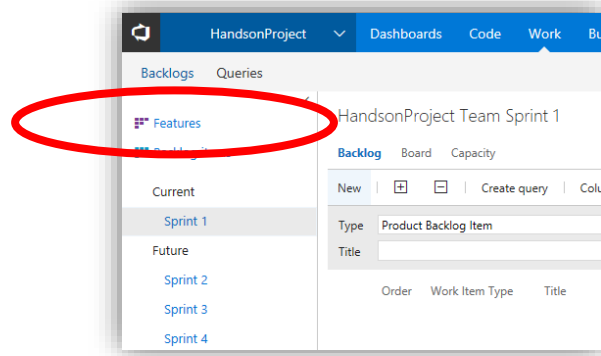
<https://vsts-handson-0001.visualstudio.com/HandsonProject>



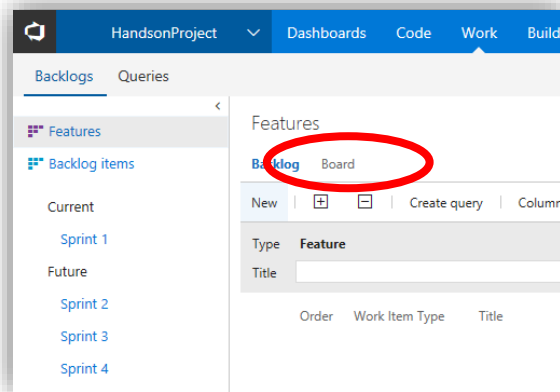
2. 画面上部のメニューバーから、[Work] – [Backlogs] をクリックします。



3. 左メニューから、[Features] をクリックします。

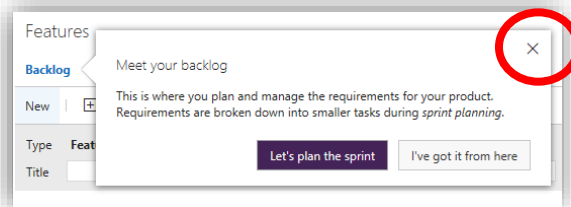


4. [Features] 画面の [Board] をクリックします。



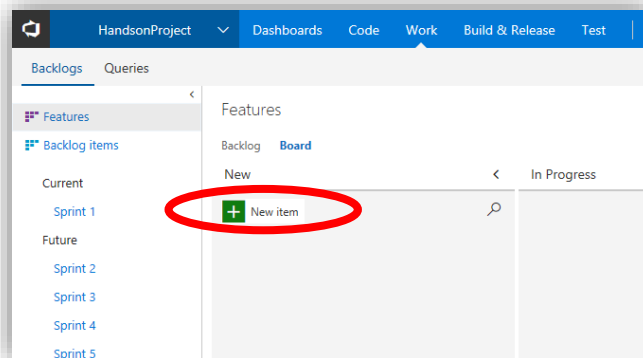
ワンポイント

初回操作時には、以下のような説明ウィンドウが表示される場合があります。このようなウィンドウが表示されたら、[×] をクリックしてウィンドウを閉じます。

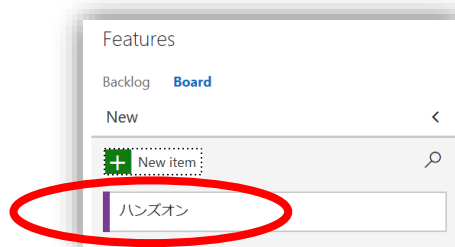


以降の操作でも、同様のウィンドウが表示された場合は、[×] をクリックして閉じてください。

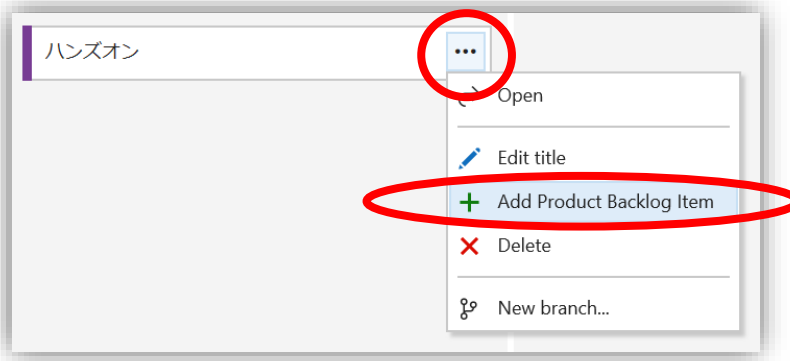
5. カンバンボードの画面に切り替わるので、[New item] をクリックします。



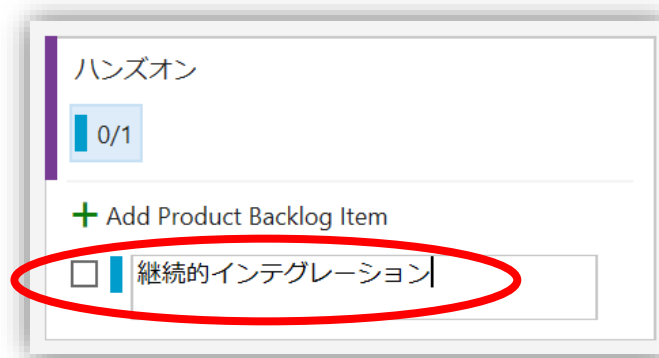
6. 新しいカンバンが作成されるので、「ハンズオン」と入力します。



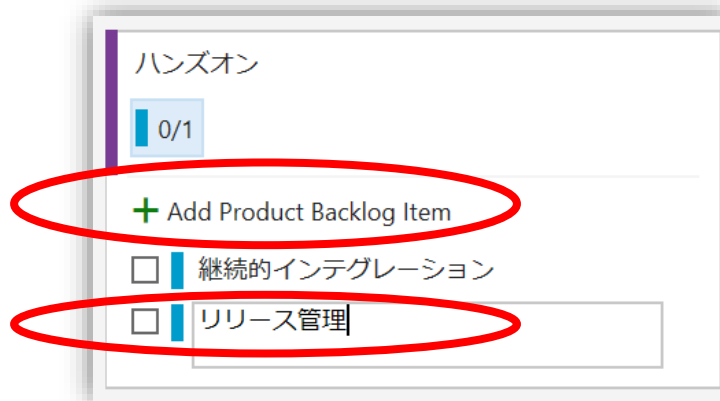
7. カンバンの右端の [...] をクリックし、[Add Product Backlog Item] をクリックします。



8. ハンズオンのカンバンの中に Backlog Item のカンバンが作成されるので、「継続的インテグレーション」と入力します。



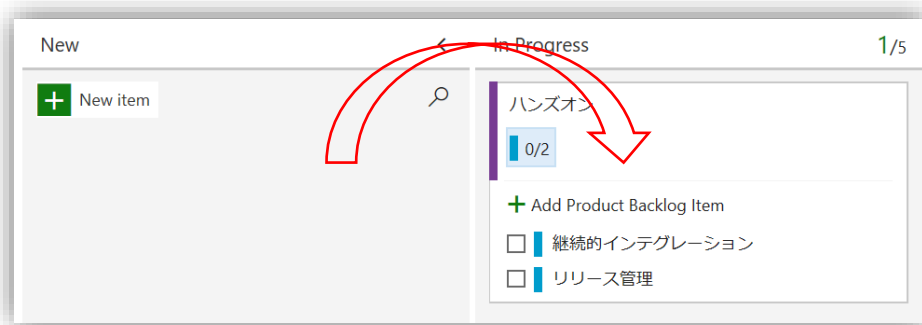
9. 引き続き、[Add Product Backlog Item] をクリックし、「リリース管理」と入力します。



ワンポイント

Product Backlog は優先順位を持っており、上にある Product Backlog ほど、作業の優先順位が高くなります。この場合では、継続的インテグレーションの作業が優先になります

10. ハンズオンのカンバンを、[In Progress] にドラッグ & ドロップします。



ワンポイント

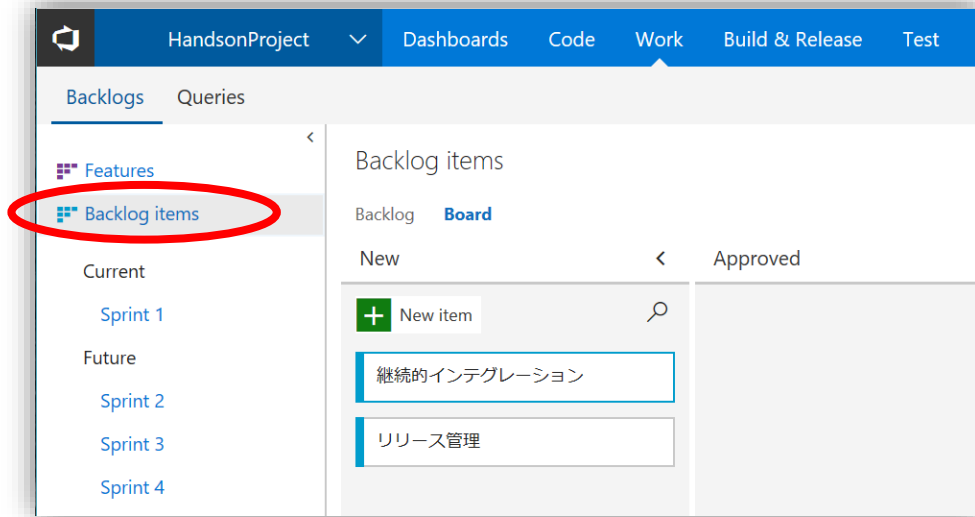
In Progress とは、「進行中である」ことを表します。Feature の作業が始まったら In Progress の状態に移行させます。

これで、Feature と Backlog Item の追加は完了です。

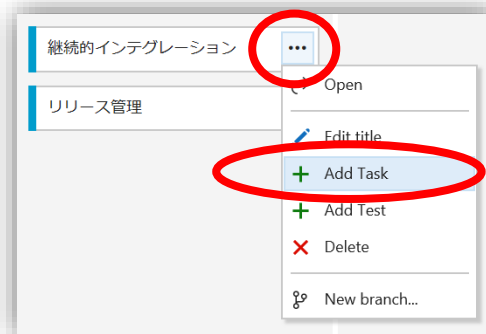
12. Task の追加

次の手順では、Backlog Item の配下に Task を追加します。

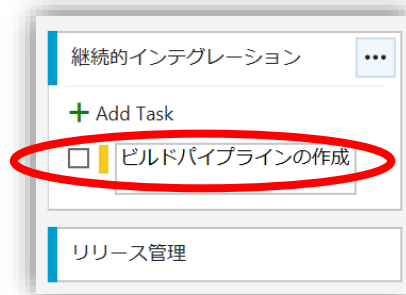
1. 引き続き、左メニューから [Backlog items] をクリックします。



2. [継続的インテグレーション] のカンバン右端の [...] をクリックし、[Add Task] をクリックします。

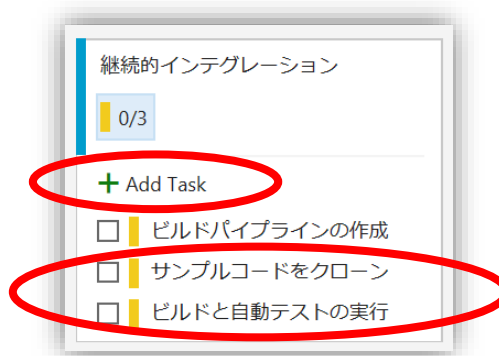


3. [継続的インテグレーション] のカンバンの中に Task のカンバンが作成されるので、「ビルドパイプラインの作成」と入力します。



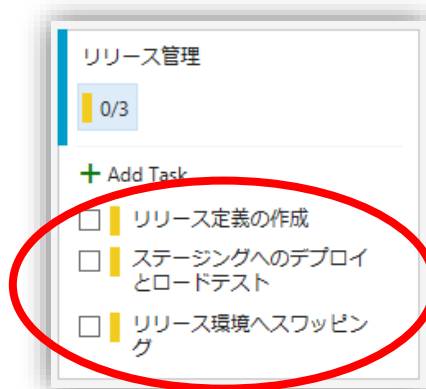
4. 引き続き、[Add Task] をクリックして以下の 2 つタスクを追加します。

- ✧ サンプルコードをクローン
- ✧ ビルドと自動テストの実行

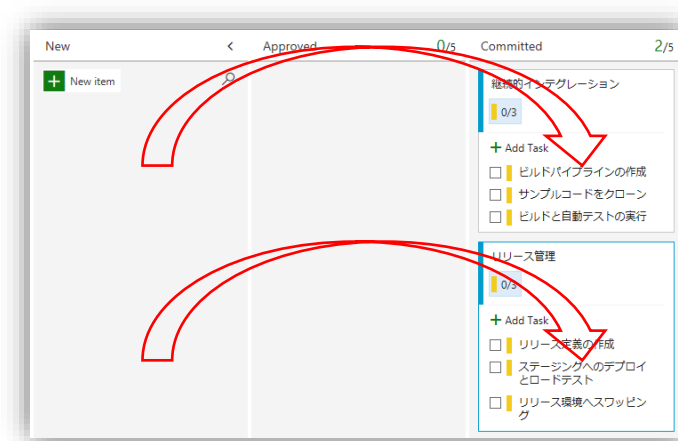


5. 同様に、[リリース管理] のカンバンにも、以下の 3 つのタスクを追加します。

- ✧ リリース定義の作成
- ✧ ステージングへのデプロイとロードテスト
- ✧ リリース環境へスワッピング



6. [継続的インテグレーション] と、[リリース管理] の Backlog Item を、[Committed] にドラッグ & ドロップします。



ワンポイント

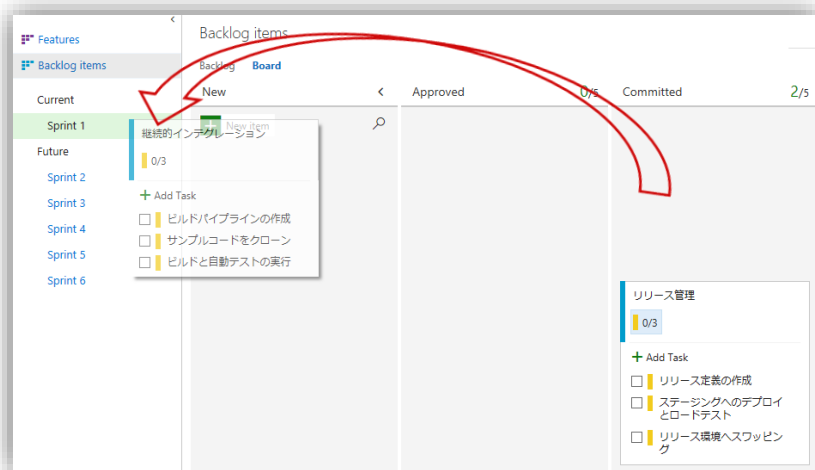
本来のスクラム開発であれば、まずはプロダクトオーナーによる **Approved** (承認) が必要になりますが、本自習書では簡易的なタスク管理を行うことを目的としているので、すぐに **Committed** (任す) にしています。

これで、**Task** の追加は完了です。

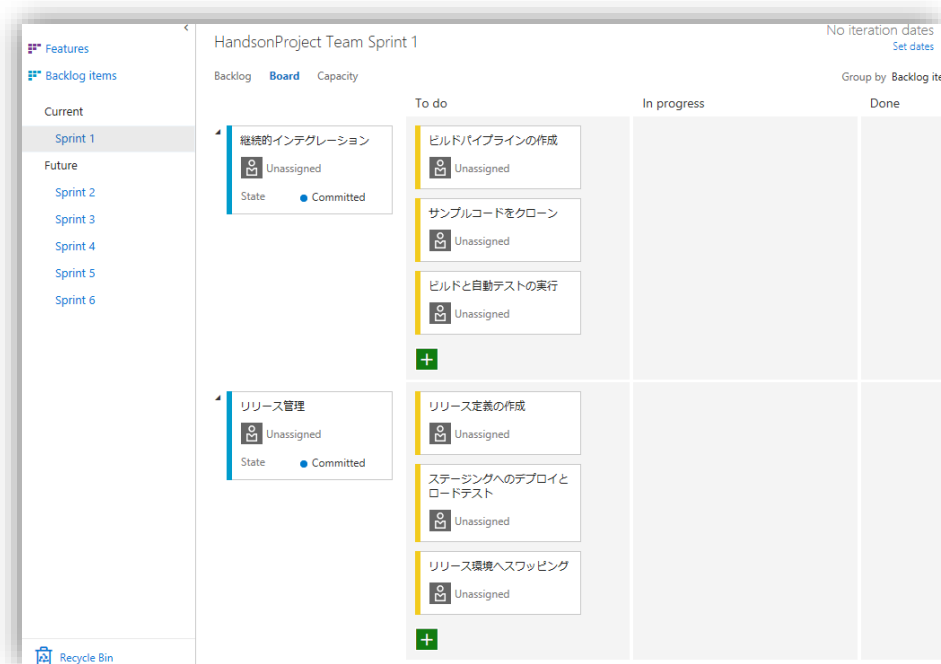
13. Sprint の開始

Sprint とは、Task を実行するためのタイムボックス (作業期間、工程) です。一般的には、1, 2 週間程度の短い期間で区切りがつくように、Sprint へ Task を配置します。次の手順では、Sprint の開始作業を行います。

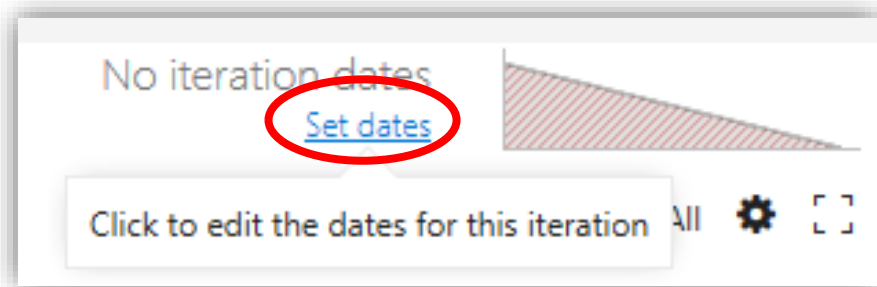
1. [継続的インテグレーション] と [リリース管理] の Backlog Item を、左メニューの [Sprint 1] にドラッグ & ドロップします。



2. 左メニューから [Sprint 1] をクリックし、Task が [To do] (行わないといけないこと) に登録されていることを確認します。

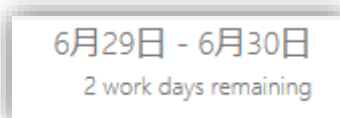


3. 次に、**Sprint** のタイムボックスを設定します。画面右上の **[Set dates]** をクリックします。



4. **[Start date]** を本日の日付、**[End date]** を次の日の日付にし、**[Save and close]** をクリックします。

5. タイムボックスが設定されます。



これで、**Sprint** の開始は完了です。**Task** の担当者割当などの詳細設定がまだ残っていますが、ここでは省略します。

STEP 3. 継続的インテグレーション (CI)

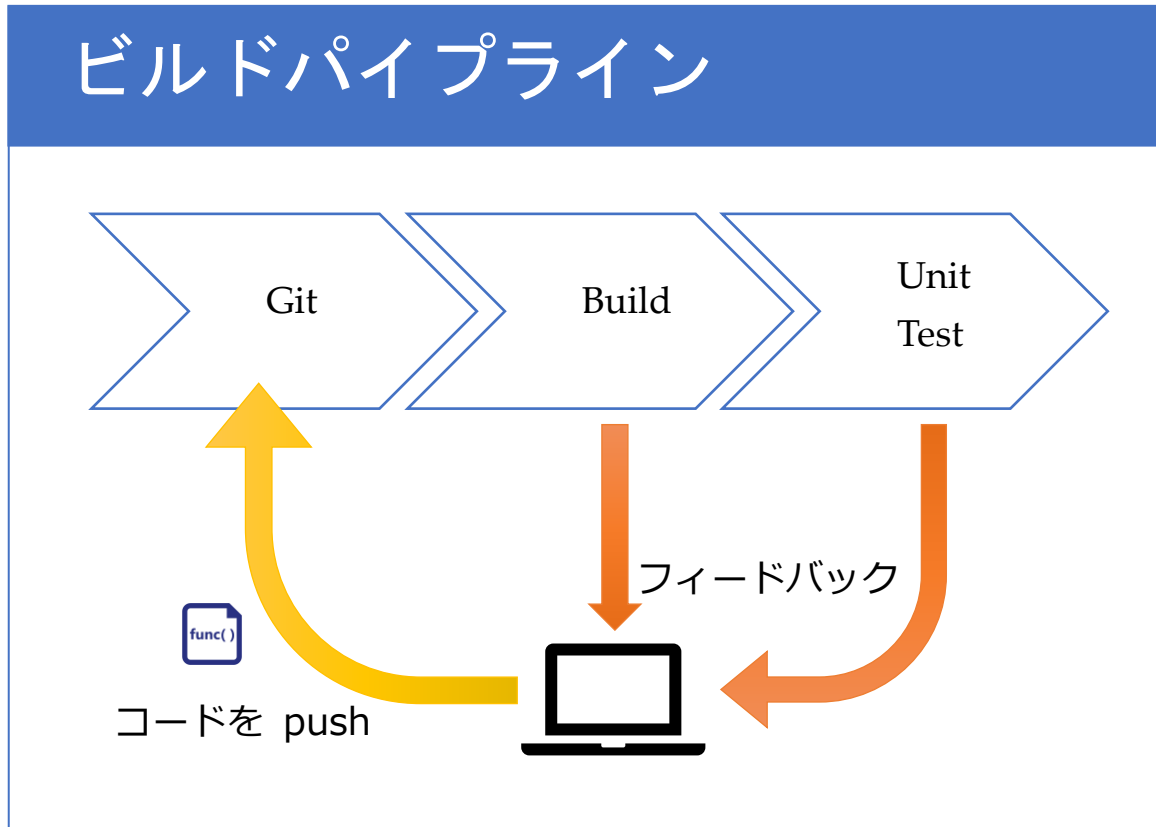
この STEP では、Visual Studio Team Services の Git リポジトリとビルドツールを使用した、継続的インテグレーションについて説明します。

この STEP では、次のことを学習します。

- ✓ ビルドパイプラインの作成
- ✓ Git リポジトリの操作
- ✓ ビルドと自動テストの実行
- ✓ ビルドエラー、またはテスト失敗時の動作確認

14. ビルドパイプラインとは

ビルドパイプライン (ビルド定義) とは、ソースコードの取得、ビルド、テストなどの複数の工程をパイプラインのように連結し、順番に実行させる仕組みです。



本自習書では、Git リポジトリに新しいコードが push されたタイミングで、最新コードでの Build → Unit Test が自動で実行されるビルドパイプラインを作成します。

また、Build と Unit Test の各工程が終了したら、成否やエラー詳細などのフィードバックを、ログやメールで取得できることも確認します。

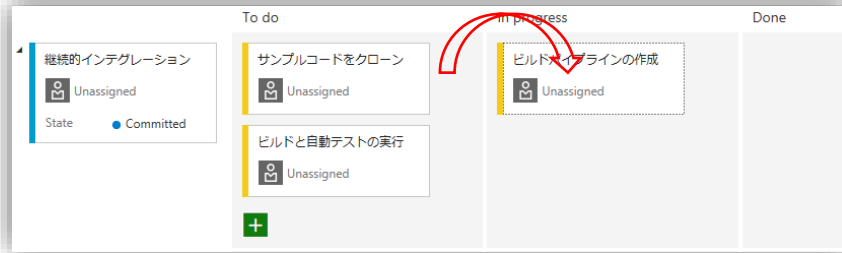
ワンポイント

Git リポジトリ、Build と Unit Test の実行環境は、すべて Visual Studio Team Services に備わっています。よって、外部サービスと連携させる必要がなく、ビルドパイプラインのすべての工程を Visual Studio Team Services だけで一元管理できます。

15. ビルドパイプラインの作成

次の手順では、ビルドパイプラインを作成します。

1. [Work] - [Backlogs] - [Sprint 1] を開きます。これから、ビルドパイプラインの作成を行うため、Task [ビルドパイプラインの作成] を [To do] から [In Progress] にドラッグ & ドロップします。



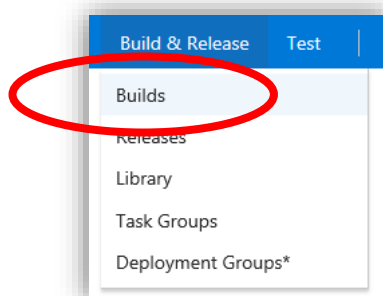
ワンポイント

Task は、作業開始前は To Do というステータスになっています。作業開始時に、In Progress にすることによって、この Task を開始したことを明示します。

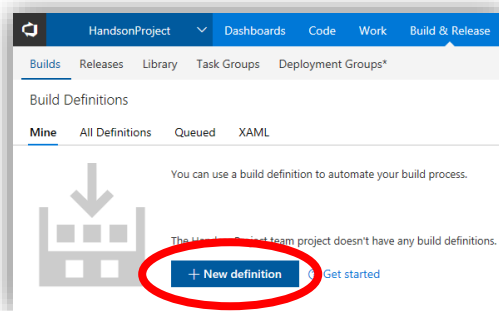
ワンポイント

Web ブラウザーのタブ機能を利用し、実習用のタブと、Backlog 用のタブで表示をわけておくと、作業がしやすくなります。

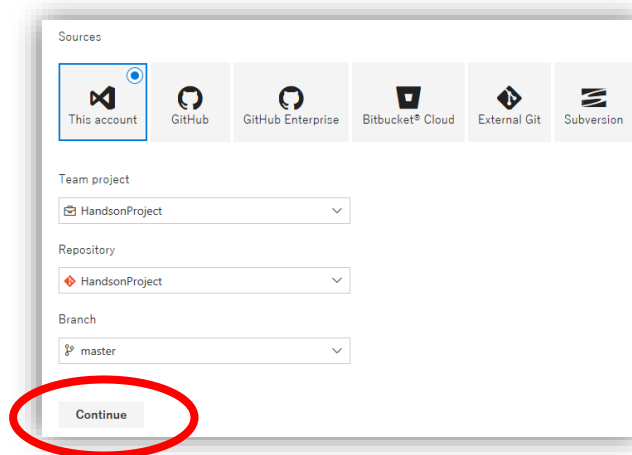
2. 画面上部のメニューバーから、[Build & Release] – [Builds] をクリックします。



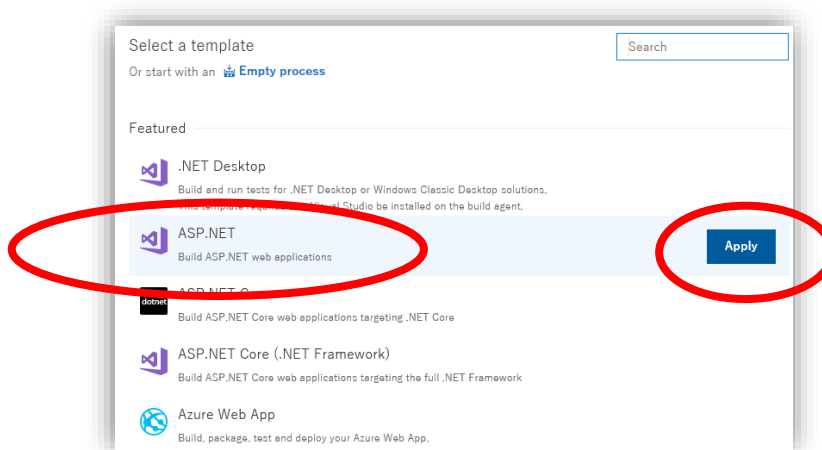
3. [Build Definitions] の画面が表示されます。ここでは、ビルドの定義 (ビルドパイプライン) を作成したり、作成済みの定義を参照したりします。
ここでは、新しいビルドパイプラインを作成するので、[New definition] をクリックします。



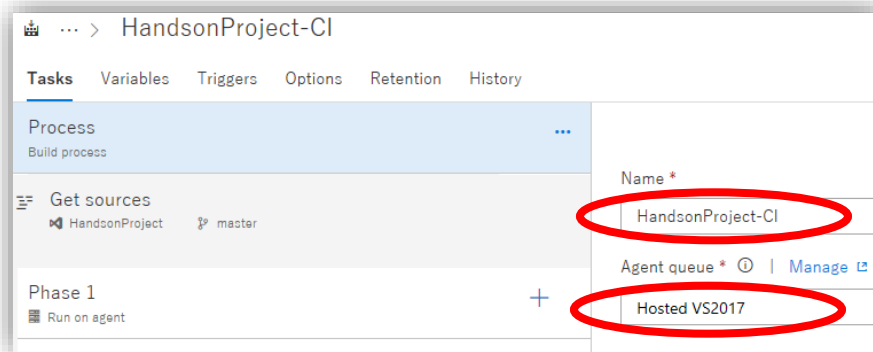
4. ソース選択画面が表示されます。本自習書では Visual Studio Team Services のプロジェクト内の Git リポジトリを使用するので、特に値を変更する必要はありません。
[Continue] をクリックします。



5. ビルドパイプラインを作成するためのテンプレート一覧画面が表示されます。デスクトップアプリや ASP.NET Web アプリなど、アプリの種類ごとにテンプレートが用意されています。
本自習書では、サンプルの Web アプリをビルドするので、[ASP.NET] を選択し、[Apply] をクリックします。

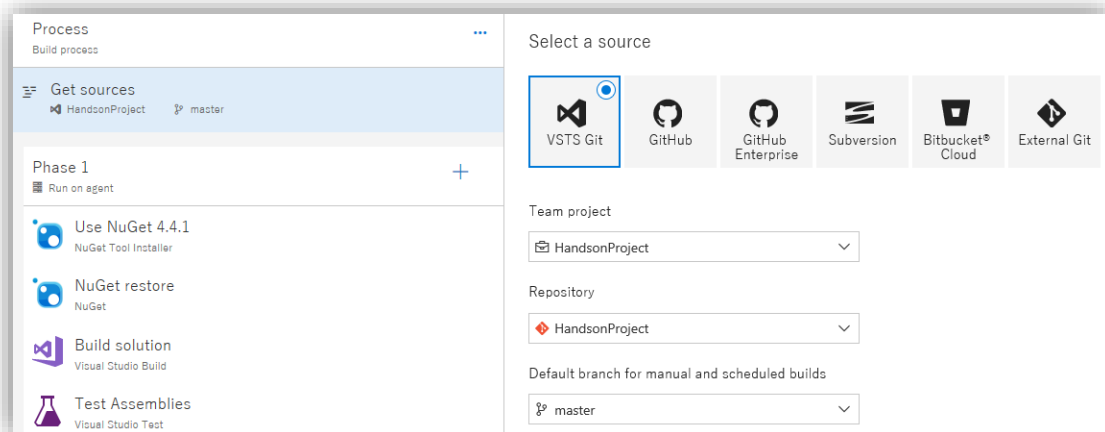


6. ASP.NET のビルドパイプラインが表示されますので、各パラメーターを以下のように入力します。



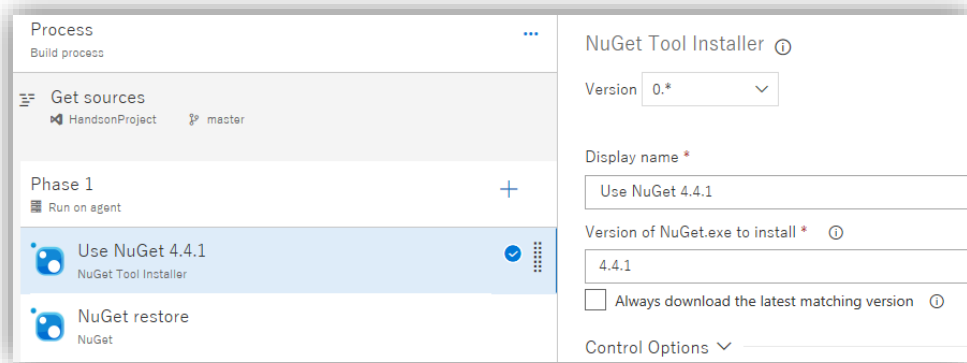
パラメーター	説明	今回の設定
Name	ビルドパイプラインの名前です。	HandsonProject-CI
Agent queue	ビルドに使用するエージェントです。	Hosted VS2017
Path to solution or packages.config	Visual Studio のソリューションファイルのパスです。	<規定値>
Artifact Name	Artifact の名前です。	<規定値>

7. プロセス一覧から、[Get Sources] をクリックします。ここでは、ソースコードをどこから取得するかを設定できますが、手順 4 で設定済みのため、ここでの操作は不要です。



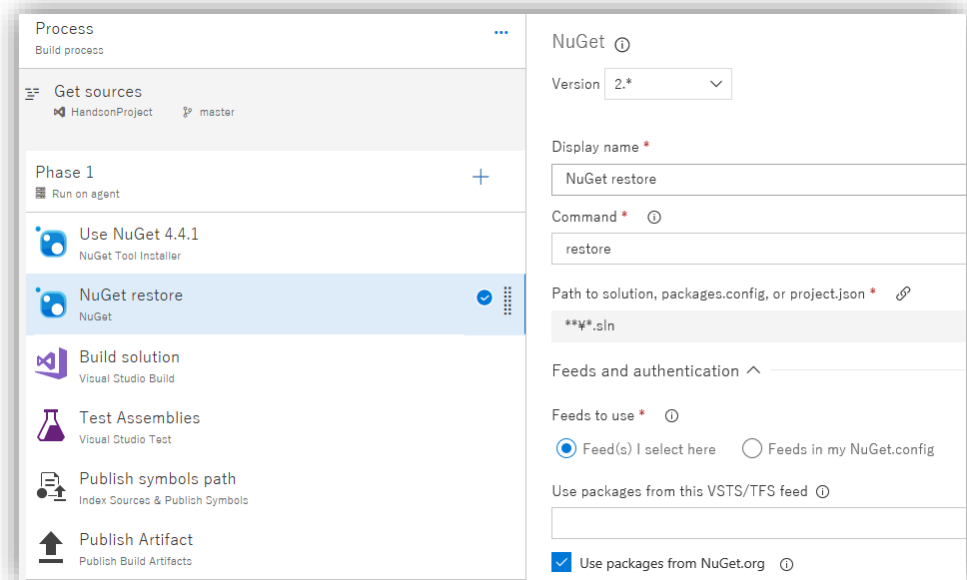
8. プロセス一覧から、[Use NuGet 4.4.1] をクリックします。ここでは、NuGet のバージョンを指定します。

この設定は、デフォルトから変更する必要ありません。



9. プロセス一覧から、[NuGet restore] をクリックします。ここでは、アプリに依存する NuGet パッケージ取得の設定をします。

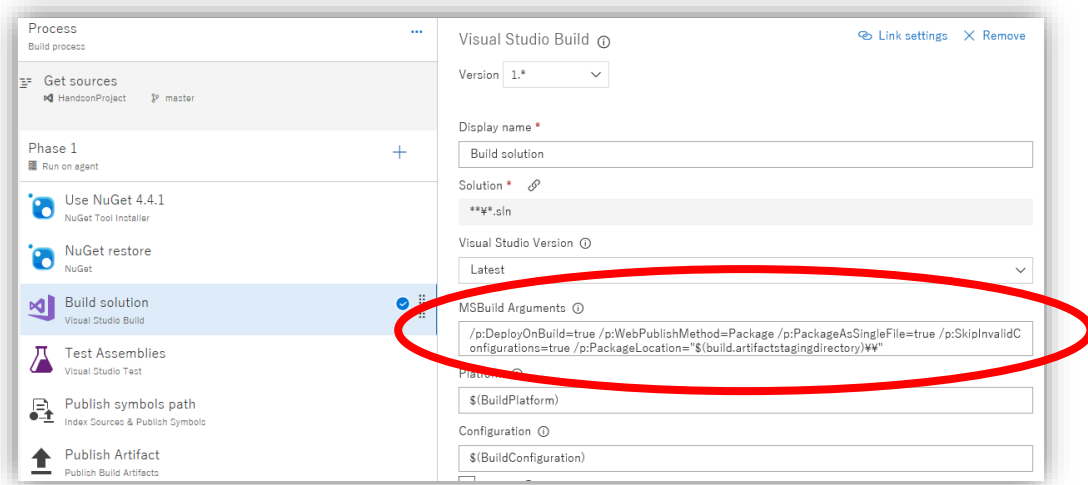
この設定は、デフォルトから変更する必要ありません。



ワンポイント

NuGet とは、.NET のパッケージマネージャーです。アプリに必要なライブラリを、NuGet リポジトリから取得することができます。詳しくは <https://www.nuget.org/> を参照してください。

10. プロセス一覧から、[Build solution] をクリックします。ここでは、アプリのビルド設定を行います。
この設定は、デフォルトから変更する必要ありませんが、[MSBuild Arguments] にビルド引数が入力されていることを確認します。



この引数は、アーティファクトを生成するために必要になります。

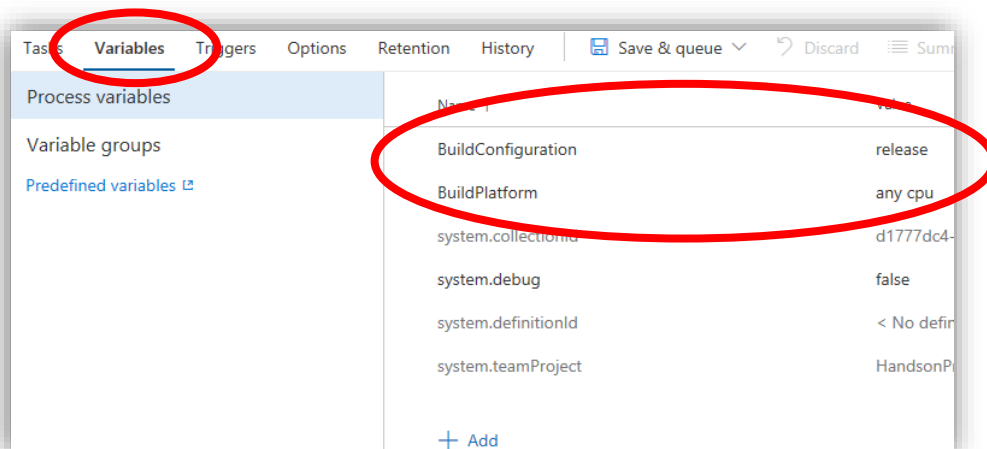


```
/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:PackageLocation="$(build.artifactstagingdirectory)¥¥"
```

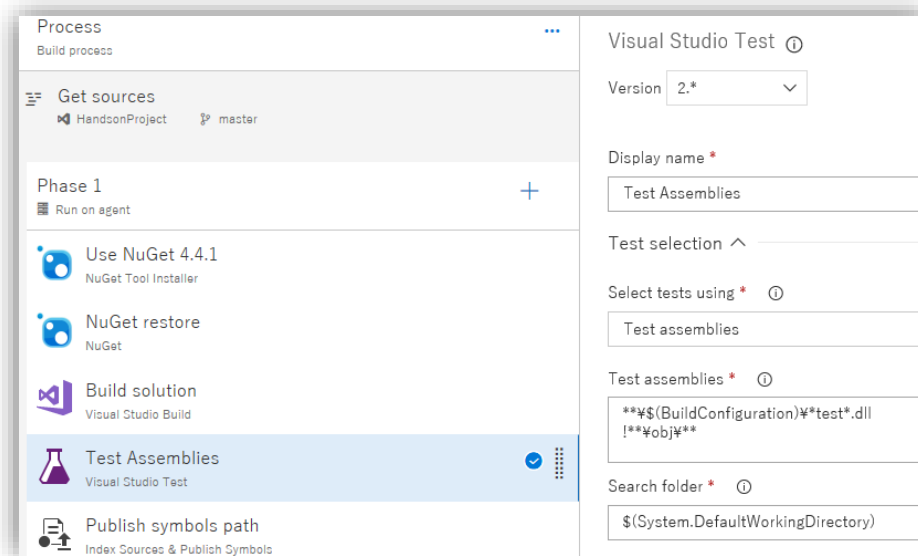
ワンポイント

パラメーター [Platform]、[Configuration] に設定されている値 [\$(...)] は変数を表します。変数は、[Variables] タブで定義されており、ここで追加、編集することができます。デフォルトでは、以下のようになっています、特に変更する必要はありません。

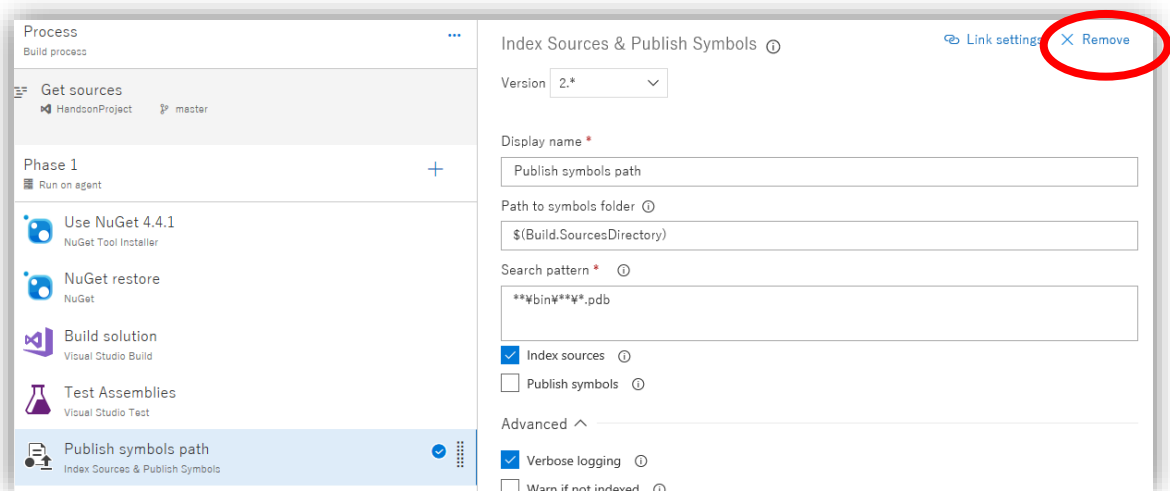
BuildPlatform: **any cpu**、

BuildConfiguration: **release**

11. プロセス一覧から、[Test Assemblies] をクリックします。ここでは、テストの実行設定を行います。本自習で使用するサンプルアプリのソリューションには、テスト (Unit Test) プロジェクトが含まれており、ビルドすることでテスト用のアセンブリが作成、実行されます。デフォルトではアセンブリを使用する設定になっているので、変更は不要です。

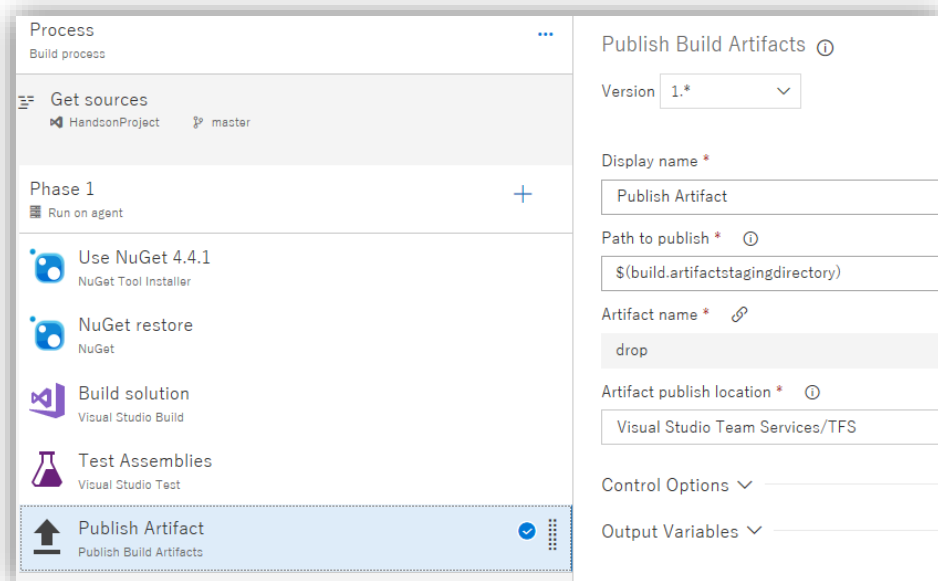


12. プロセス一覧から、[Publish symbols path] をクリックします。本工程は不要なので、[Remove] をクリックして削除します。



13. プロセス一覧から、[Publish Artifact] をクリックします。ここでは、Artifact の名前や保存場所などの設定を行います。

この設定は、デフォルトから変更する必要ありません。

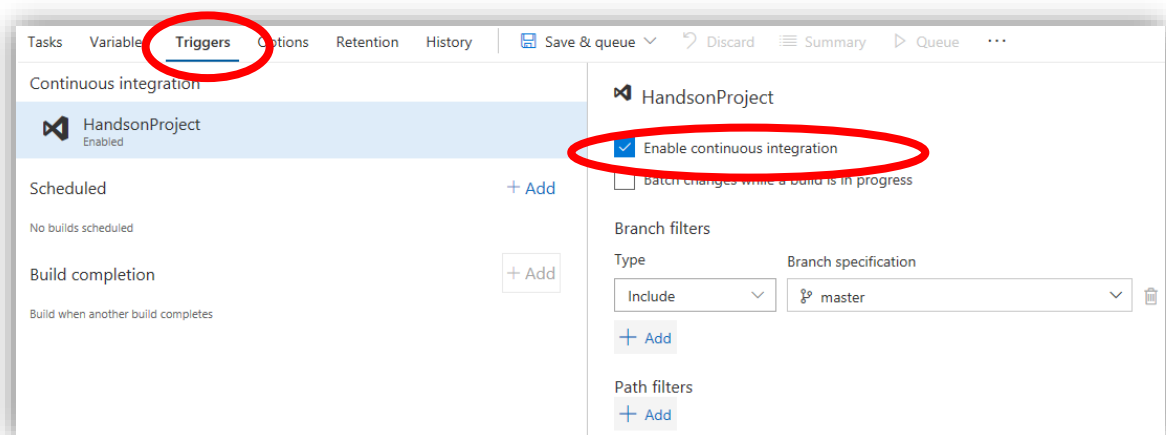


ワンポイント

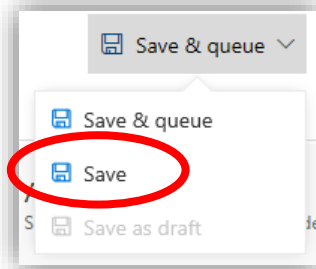
Artifact とはビルド成果物です。Web アプリでは、画像、スクリプト、CSS、Web ページ などのリソースファイルや、アプリを動かすアセンブリファイルなどになります。

14. [Triggers] タブをクリックします。この画面では、ビルドパイプラインが実行されるタイミングを設定します。

本自習書では、Git リポジトリにコードが push されたタイミングでこのビルドパイプラインを実行するので、[Enable continuous integration] をチェックします。



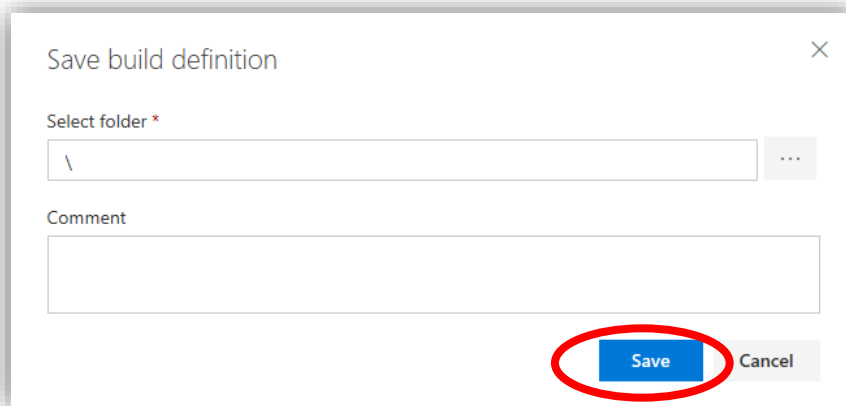
15. これでビルドパイプラインが完成したので、[Save & queue] – [Save] をクリックします。



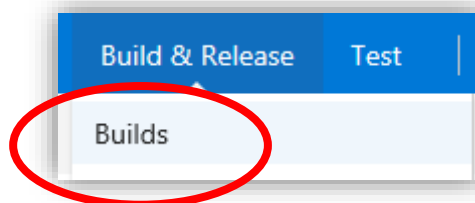
ワンポイント

Save & queue をクリックすると、すぐにビルドパイプラインを実行することができますが、現時点では Git リポジトリは空なので、Save だけ行います。

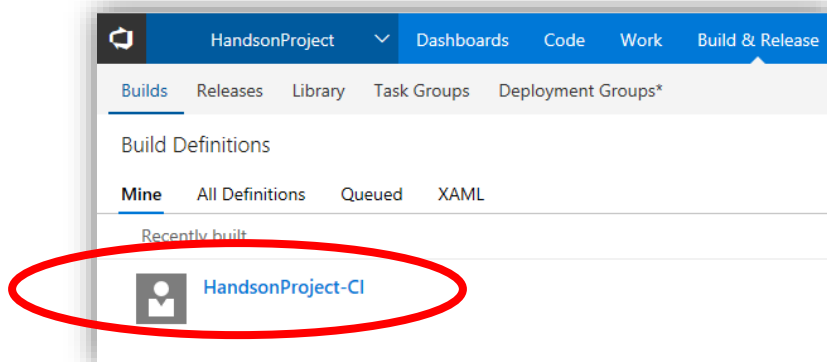
16. 保存場所はデフォルトで問題ないので、[Save] をクリックします。



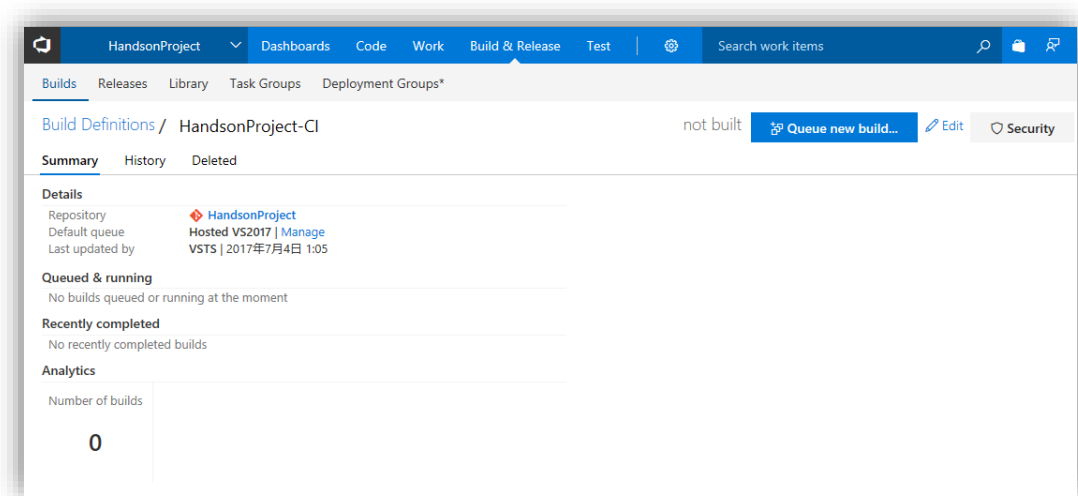
17. メニューバーから、[Build & Release] – [Builds] をクリックします。



18. [HandsonProject-CI] という名前のビルドパイプラインが作成されていることを確認します。



19. この [HandsonProject-CI] をクリックします。ビルドパイプラインの Summary が表示されることを確認します。



20. 最後に、[Work] - [Backlogs] - [Sprint 1] を開きます。カンバンボードが表示されない場合は、画面左から [Sprint 1] をクリックし、[Board] をクリックします。
これでビルドパイプラインの作成は完了のため、Task [ビルドパイプラインの作成] を [In Progress] から [Done] にドラッグ & ドロップします。



ワンポイント

Task 完了後は、ステータスを In Progress から Done にします。

これで、ビルドパイプラインの作成は完了です。

16. サンプルコードの clone と Git の設定

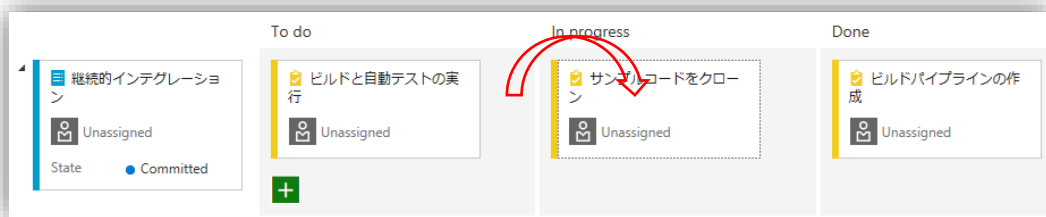
Visual Studio Team Services にビルドパイプラインが完成しました。続いて、このビルドパイプラインでビルドするためのサンプルアプリのコードを作業 PC 上にダウンロードします。

次の手順では、Visual Studio Code と git コマンドを使用して、Github にあるサンプルアプリのコードを clone します。

ワンポイント

clone とは、リモートリポジトリをそのままローカルの開発環境へ複製 (ダウンロード) することです。Github は、リモートリポジトリを提供する代表的なサービスです。

1. Visual Studio Team Services の [Work] - [Backlogs] - [Sprint 1] を開きます。これから、サンプルコードの clone を行うため、Task [サンプルコードのクローン] を [To do] から [In Progress] にドラッグ & ドロップします。

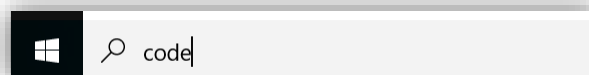


2. Visual Studio Code を起動します。

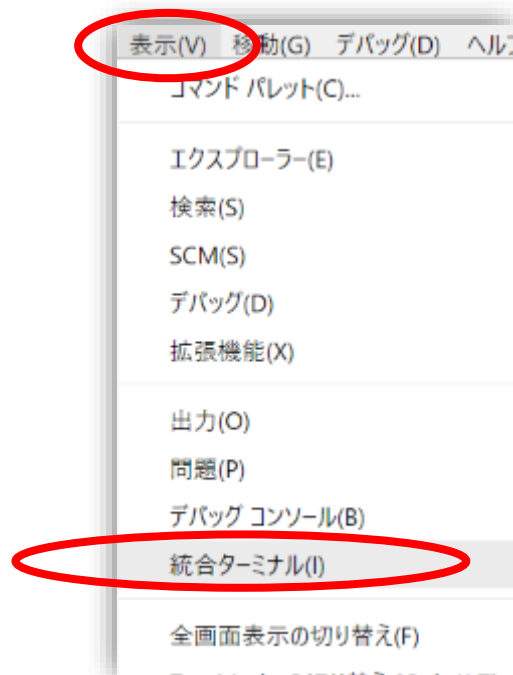


ワンポイント

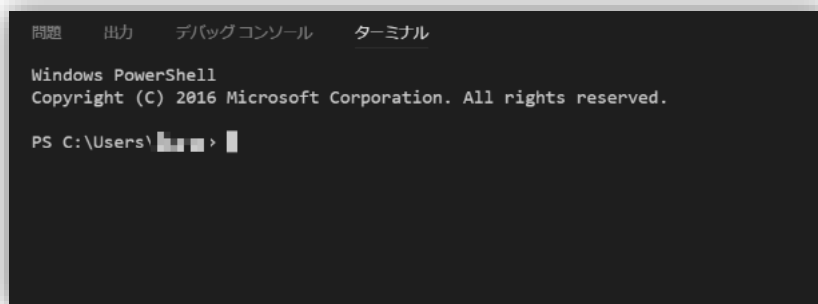
Windows 10 の場合は、Cortana から 「code」 と入力すると簡単に起動できます。



3. メニューバーの [表示] - [統合ターミナル] をクリックします。

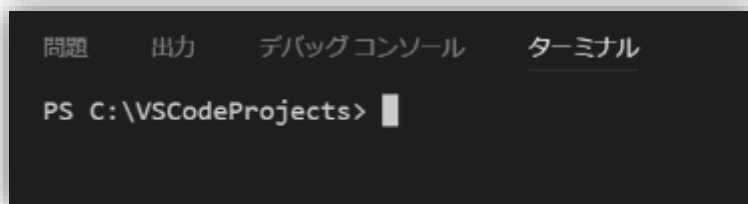


4. ターミナルが表示されます。



5. 以下のコマンド実行し、リポジトリを clone するためのフォルダーを作成します。フォルダーのパスは「C:¥VSCodeProjects」にし、作成後このフォルダーに移動します。
※macOS の場合は、適当な場所に VSCodeProjects フォルダーを作成してください。

```
mkdir C:¥VSCodeProjects
cd C:¥VSCodeProjects¥
```



6. 以下のコマンド実行し、Github からリポジトリを clone します。

```
git clone https://github.com/creationline/vsts-handson-webapp.git
```

```
PS C:\VSCodeProjects> git clone https://github.com/creationline/vsts-handson-webapp.git
Cloning into 'vsts-handson-webapp'...
remote: Counting objects: 43, done.
remote: Total 43 (delta 0), reused 0 (delta 0), pack-reused 43
Unpacking objects: 100% (43/43), done.
PS C:\VSCodeProjects>
```

7. 以下のコマンドを実行し、clone されたフォルダー [vsts-handson-webapp] に移動します。このフォルダーが、サンプルアプリのリポジトリのトップフォルダーになります。

```
cd vsts-handson-webapp
```

```
問題 出力 デバッグ コンソール ターミナル

PS C:\VSCodeProjects> cd vsts-handson-webapp
PS C:\VSCodeProjects\vsts-handson-webapp>
```

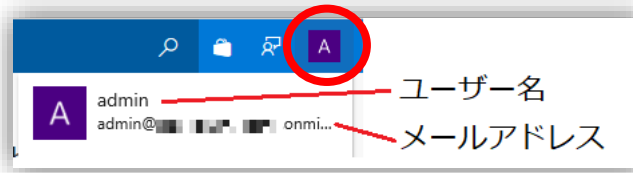
8. 以下のコマンドを実行し、リポジトリの内容を確認します。

```
tree /F
```

```
C:\.
├── .gitattributes
├── .gitignore
├── HandsOnWebApp.sln
├── README.md
├── HandsOnWebApp
│   ├── Default.aspx
│   ├── Default.aspx.cs
│   ├── Default.aspx.designer.cs
│   ├── Error.aspx
│   ├── Error.aspx.cs
│   ├── Error.aspx.designer.cs
│   ├── Global.asax
│   ├── Global.asax.cs
│   ├── HandsOnWebApp.csproj
│   ├── packages.config
│   ├── Web.config
│   ├── Web.Debug.config
│   └── Web.Release.config
├── Properties
│   └── AssemblyInfo.cs
├── HandsOnWebApp.Tests
│   ├── HandsOnWebApp.Tests.csproj
│   └── UnitTest.cs
└── Properties
    └── AssemblyInfo.cs
```

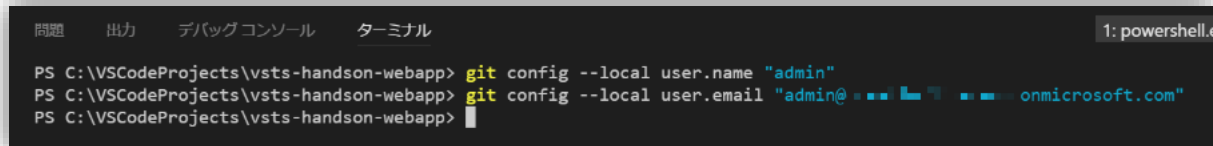

9. これで、サンプルアプリの `clone` が完了しました。

引き続き、Git のユーザー設定を行います。Visual Studio Team Services の画面右上のプロフィール画像にマウスカーソルを乗せると、ユーザー名とメールアドレスが表示されますので、これを控えておきます。



10. Visual Studio Code のターミナルで以下のコマンドを実行し、Git リポジトリに対して、先ほど控えていたユーザー名とメールアドレスを設定します。

```
git config --local user.name "<ユーザー名>"
git config --local user.email "<メールアドレス>"
```

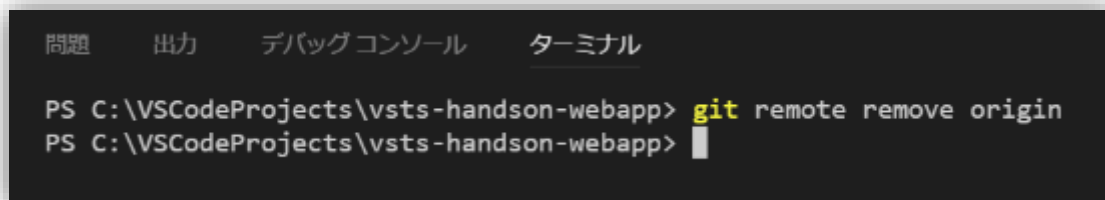


ワンポイント

Git では、リモートリポジトリへ `push` (ローカル環境で編集したコードをリモートへ持っていく) する際に、誰が `push` したかわかるように、ユーザー名とメールアドレスの設定が必要になります。

11. 引き続き、リモートリポジトリの切り替えを行います。以下のコマンドを実行し、リモートリポジトリから GitHub を削除します。

```
git remote remove origin
```



ワンポイント

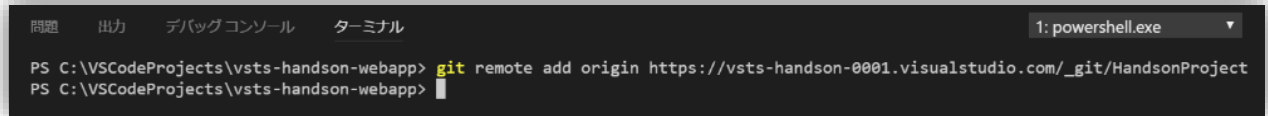
`clone` が終わったばかりのこの時点では、GitHub がリモートリポジトリとして設定されています。以後は、Visual Studio Team Services のプロジェクトが持つリポジトリにコードを `push` していくので、リモートリポジトリを変更する必要があります。

12. 以下のコマンドを実行し、リモートリポジトリを Visual Studio Team Services のリポジトリにします。

※ドメイン が **vsts-handson-0001** の場合、Git リポジトリは以下の URL になります。Visual Studio Team Services サインアップ時に指定したドメインに置き換えてください。

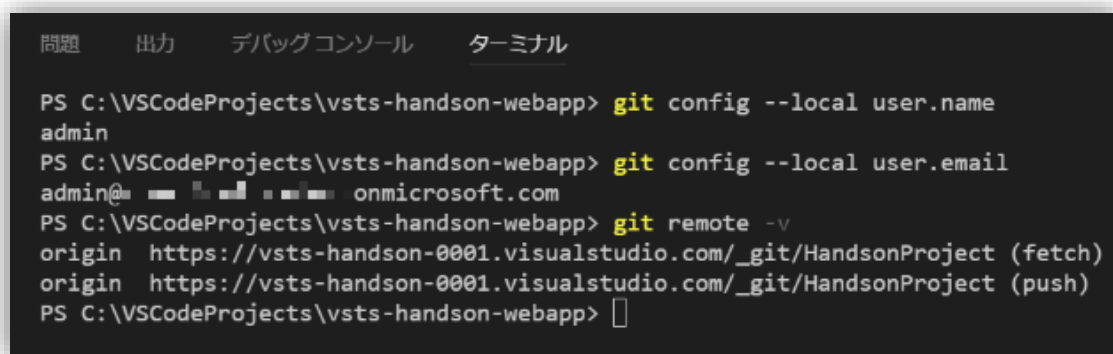
https://vsts-handson-0001.visualstudio.com/_git/HandsonProject

```
git remote add origin https://vsts-handson-0001.visualstudio.com/_git/HandsonProject
```



13. 以下のコマンドを実行し、これまで設定した項目を確認します。コマンド実行後、それぞれ設定項目が表示されます。

```
git config --local user.name
git config --local user.email
git remote -v
```



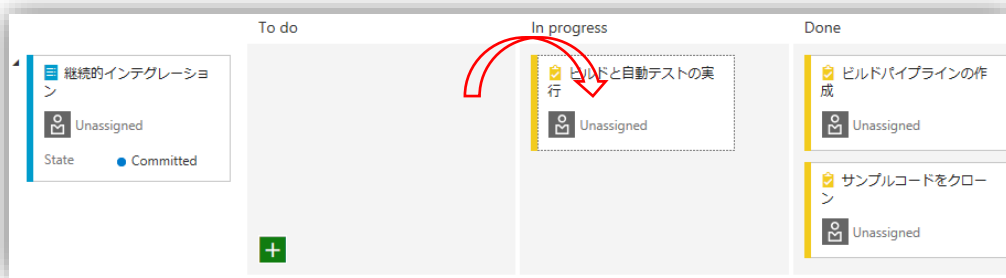
14. 最後に、Visual Studio Team Services の [Work] - [Backlogs] - [Sprint 1] を開きます。これでサンプルコードの clone は完了のため、Task [サンプルコードをクローン] を [In Progress] から [Done] にドラッグ & ドロップします。



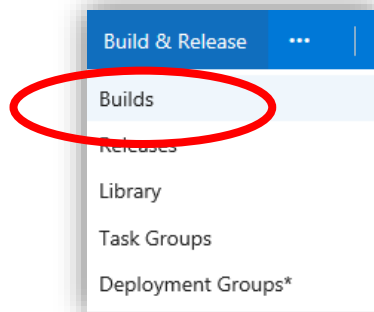
17. ビルドと自動テストの実行

次の手順では、Visual Studio Code を使用して、サンプルアプリのコードを Visual Studio Team Services の Git リポジトリに push します。また、このタイミングでビルドパイプラインが自動で実行されることを確認します。

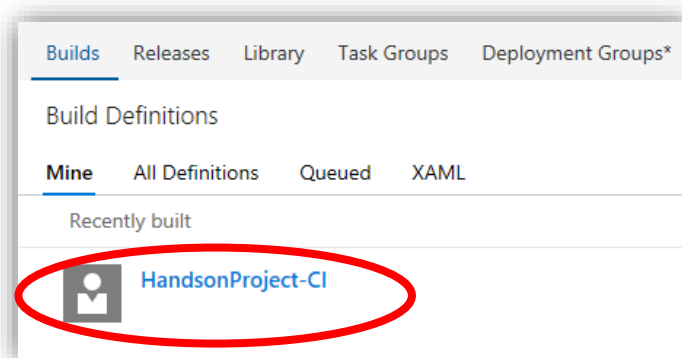
1. Visual Studio Team Services の [Work] - [Backlogs] - [Sprint 1] を開きます。これから、ビルドと自動テストの実行を行うため、Task [ビルドと自動テストの実行] を [To do] から [In Progress] にドラッグ & ドロップします。



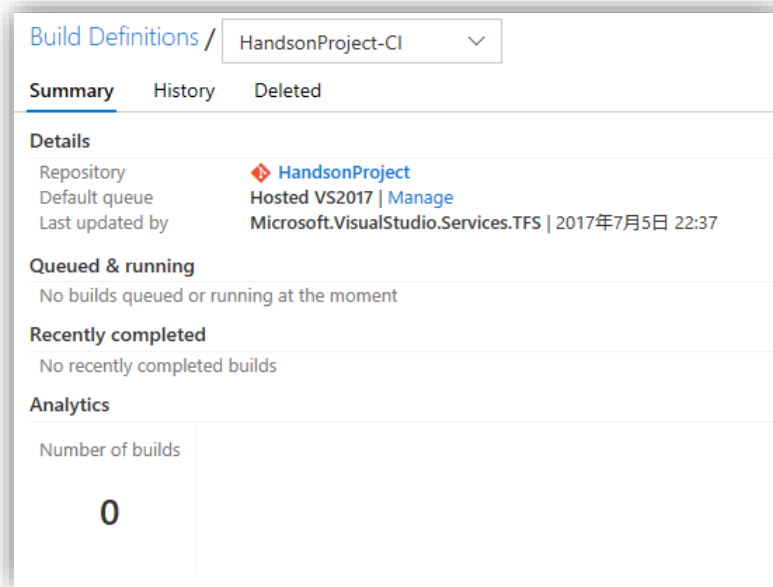
2. ビルドパイプラインが自動で実行されることを確認するために、ビルドパイプラインの Summary 画面を表示しておきます。メニューバーから、[Build & Release] - [Builds] をクリックします。



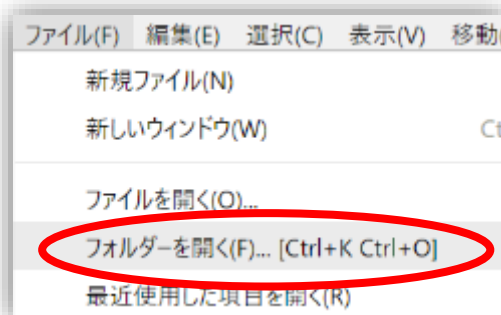
3. [Builds Definitions] から、[HandsonProject-CI] をクリックします。



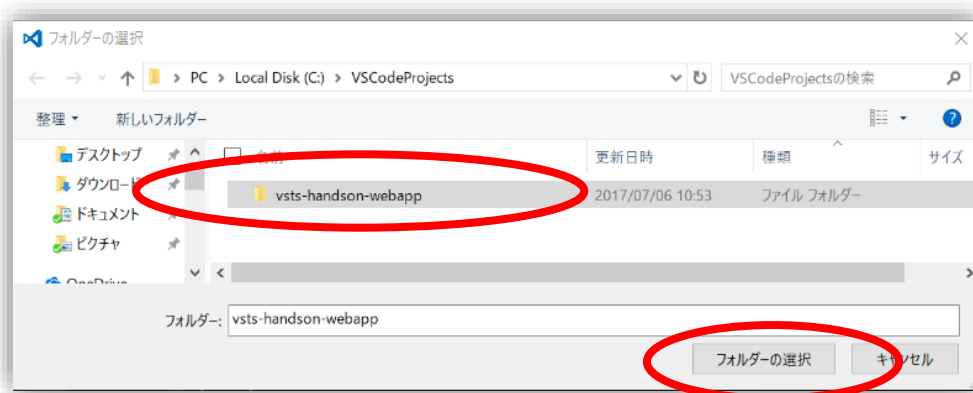
4. ビルドパイプライン [HandsonProject-CI] の Summary 画面が表示されます。これ以降の作業は、この Summary 画面を表示させたまま Visual Studio Code の操作を行います。



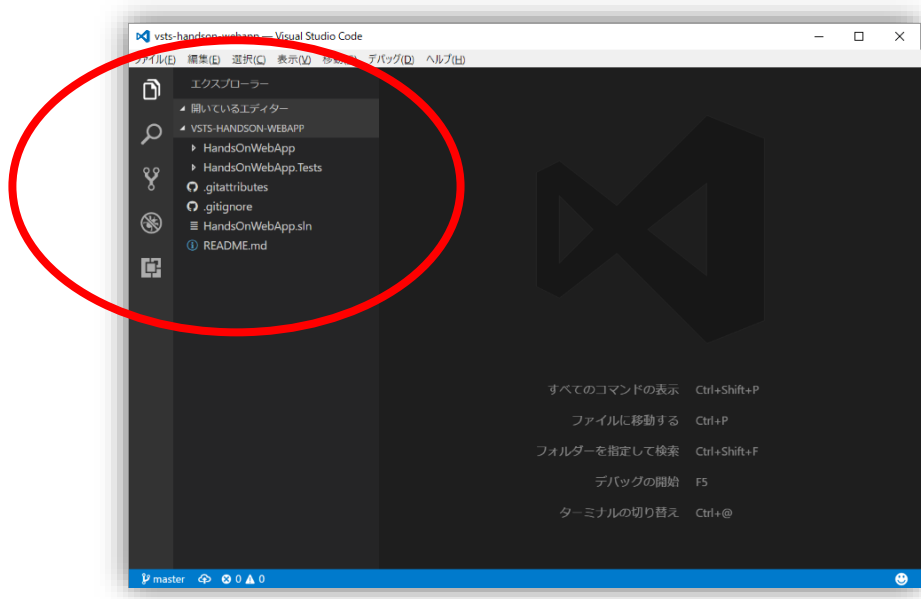
5. Visual Studio Code のメニューバーから、[ファイル] - [フォルダーを開く] をクリックします。



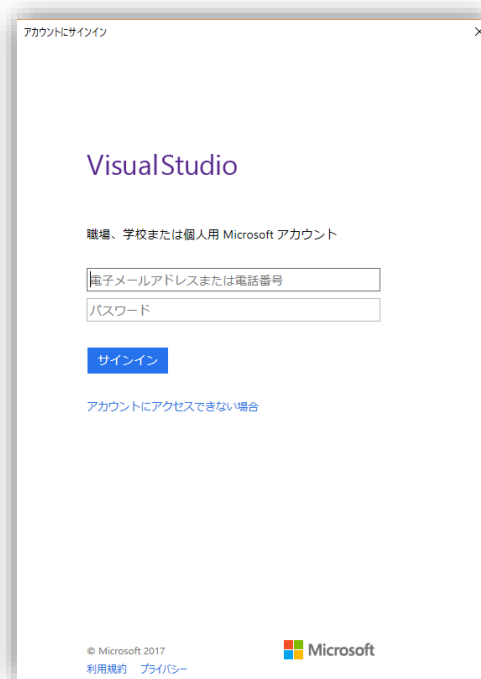
6. [フォルダーの選択] ダイアログが表示されるので、先ほど clone した [vsts-handson-webapp] を選択し、[フォルダーの選択] をクリックします。



7. [vsts-handson-webapp] 配下のファイルやフォルダーが、[エクスプローラー] に表示されます。



8. 初回実行時は Visual Studio Team Services へのサインインを要求されます。Visual Studio Team Services へのサインアップで使った Microsoft アカウントのメールアドレスとパスワードを入力して、[サインイン] をクリックします。



9.

ワンポイント

Windows の場合、資格情報マネージャーによりこのアカウントが管理されるため、2 回目以降はサインインする必要ありません。

資格情報は、コマンドプロンプトから「cmdkey /list」を実行することで確認できます。

10. メニューバーから、[表示] - [統合ターミナル] をクリックします。



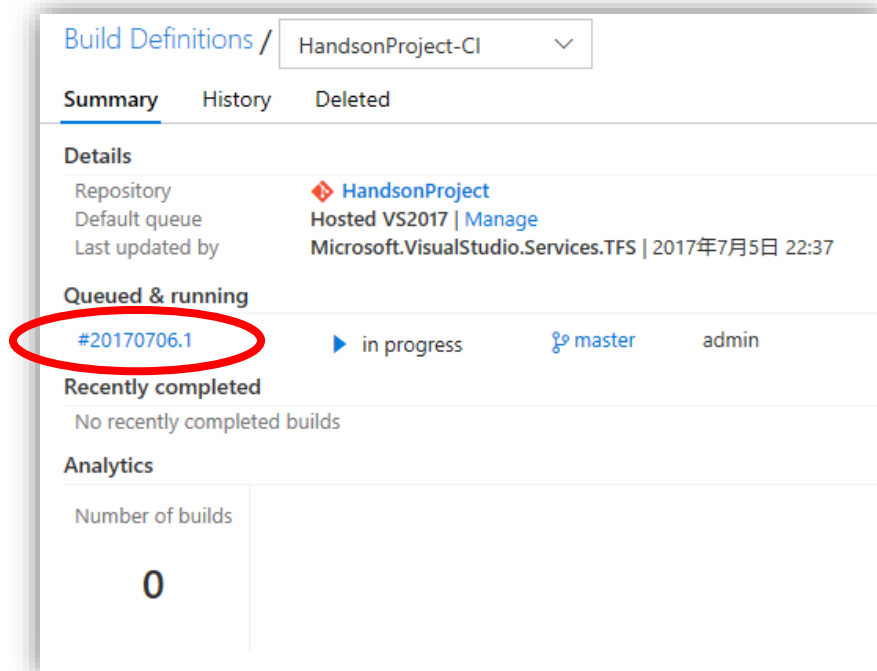
11. ターミナルから以下のコマンドを実行して、Visual Studio Team Services の Git リポジトリに最初の push を行います。

```
git push -u origin --all
```

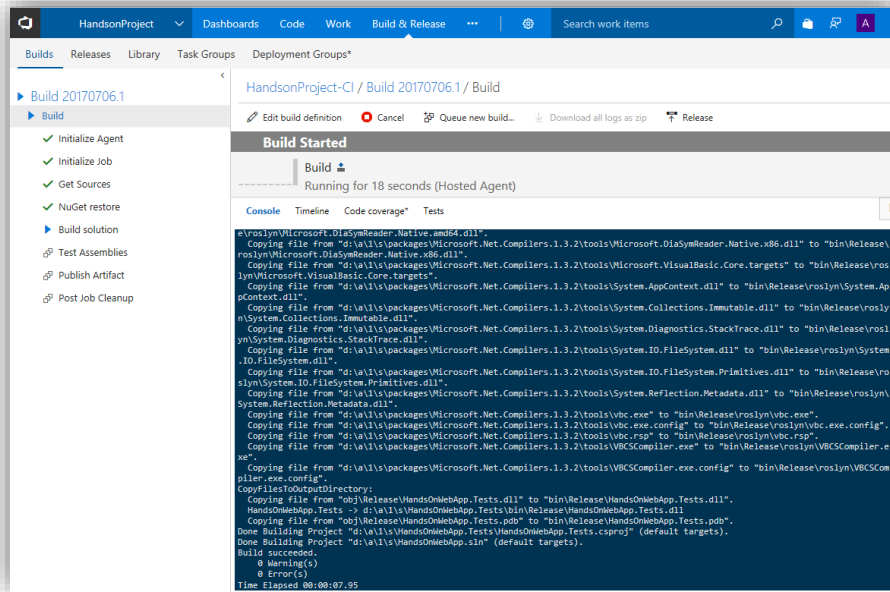
12. ローカルリポジトリがそのまま Visual Studio Team Services の Git リポジトリへ push されます。

```
PS C:\VSCodeProjects\vsts-handson-webapp> git push -u origin --all
Counting objects: 43, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (40/40), done.
Writing objects: 100% (43/43), 14.87 KiB | 2.97 MiB/s, done.
Total 43 (delta 15), reused 0 (delta 0)
remote: Analyzing objects... (43/43) (7 ms)
remote: Storing packfile... done (72 ms)
remote: Storing index... done (70 ms)
To https://vsts-handson-0001.visualstudio.com/_git/HandsonProject
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
PS C:\VSCodeProjects\vsts-handson-webapp> 
```

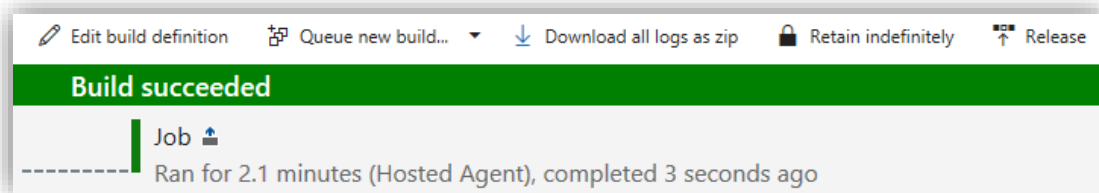
13. push 完了のタイミングで、Visual Studio Team Services の ビルドパイプラインの Summary 画面では、[Queued & running] からビルドパイプラインが開始されたことを確認できます。
詳細を確認するために、ビルドパイプラインキューの日付 (以下の画像では [#20170706.1]) をクリックします。



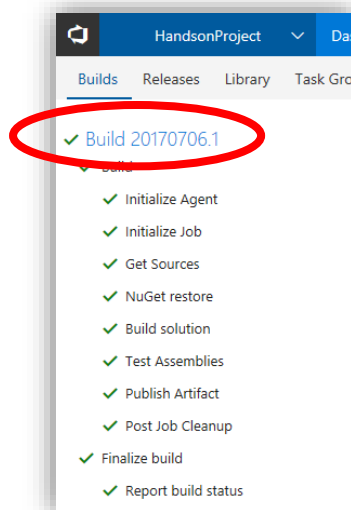
14. 現在進行中のビルドパイプラインのコンソール出力が表示されます。ビルドの進行状況をリアルタイムで確認できます。



しばらくしてビルドが完了し、[Build Succeeded] が表示されることを確認します。



15. 画面左に表示されるビルドプロセスの一番上のリンク（以下の画像では [Build 20170706.1]）をクリックします。



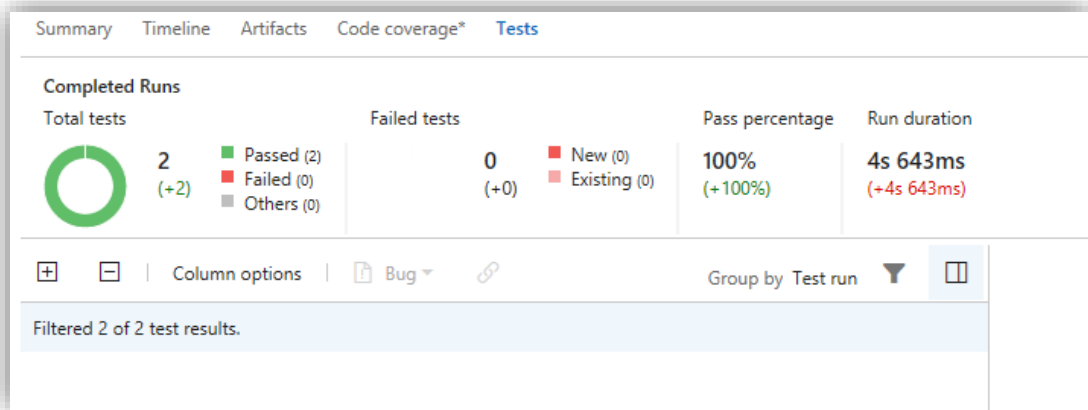
16. ビルド結果の **Summary** が表示されます。ここでは、ビルドの実行日時やテスト結果などが表示されます。

The screenshot displays the 'Build Summary' page for 'HandsonProject-CI / Build 20170706.1'. The build status is 'Build succeeded'. The summary includes build details such as Definition (HandsonProject-CI), Source (master), and Source version (Commit e04853d0). It also shows test results: 2 tests passed, 0 failed, and 0 others. The build duration is 936ms. The page includes tabs for Summary, Timeline, Artifacts, Code coverage*, and Tests. The 'Associated changes' section lists two commits: e04853d and 18ada71. The 'Work items linked to associated changes' section shows no work items. The 'Code Coverage' section indicates no build code coverage data is available. The 'Tags' section has an 'Add tag...' button. The 'Deployments' section shows no deployments found for this build.

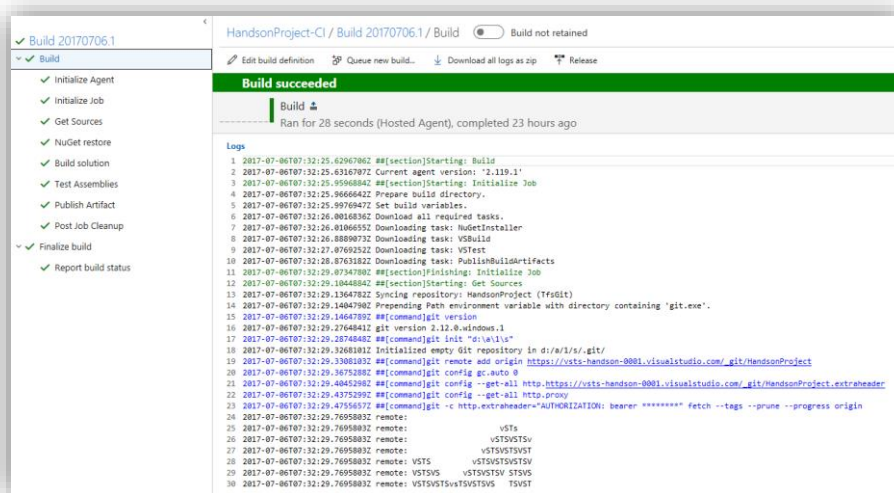
17. [Artifacts] タブをクリックすると、Artifact (ビルド成果物) のダウンロードや中身の確認ができます。

The screenshot shows the 'Artifacts' tab in the Azure DevOps interface. The 'Artifacts' tab is highlighted with a red circle. Below the tabs, there is a table with columns for 'Name', 'Download', and 'Explore'. A single artifact named 'drop' is listed under the 'Name' column. The 'Download' and 'Explore' buttons are visible next to the artifact name.

18. [Tests] タブをクリックすると、テスト (Unit Test) の結果が表示されます。以下の結果では、2つのテスト項目が問題なく全てパスしたことを確認できます。



19. 画面左側のビルドパイプラインの各プロセスをクリックすると、それぞれのログを表示することができます。ここでは、主なプロセスの役割を確認します。



● Initialize Agent

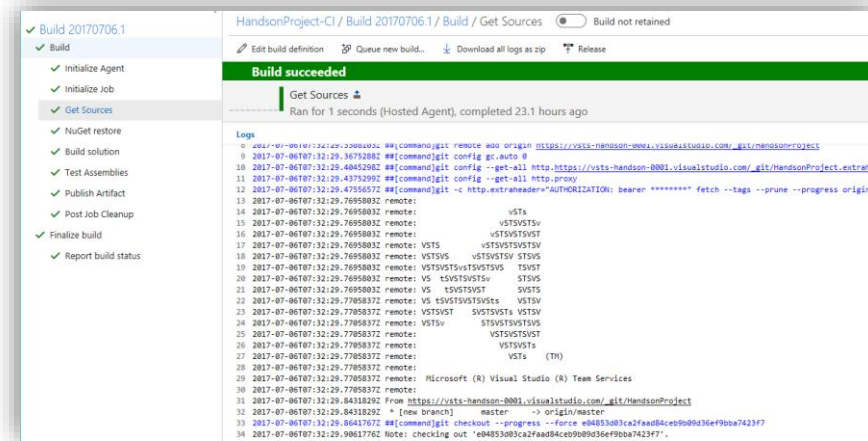
ホストされた Agent が、ビルドを実行する準備をします。

● Initialize Job

Agent が Job (ビルドパイプライン) を実行する準備をします。

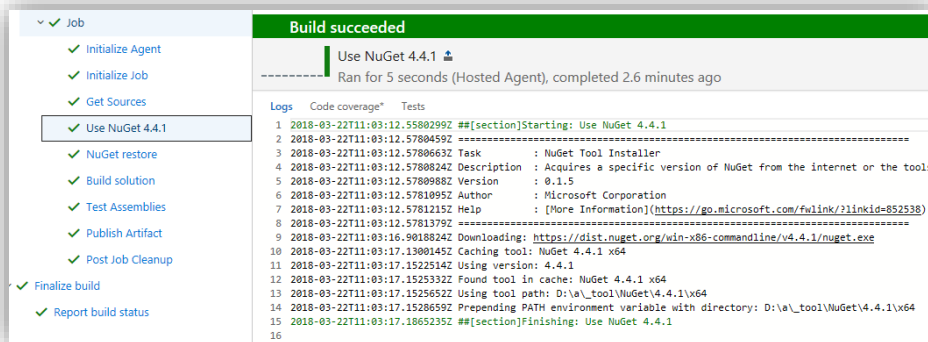
●Get Sources

Git リポジトリからコードを取得します。



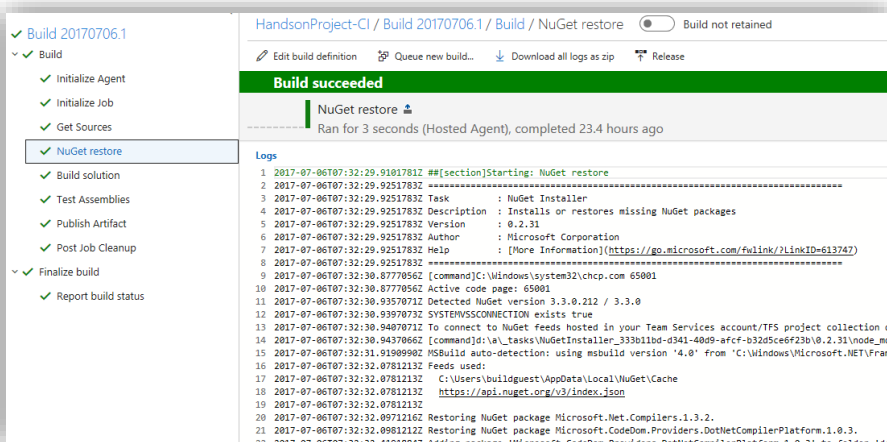
●Use NuGet 4.4.1

NuGet Version 4.4.1 を使用する準備をします。



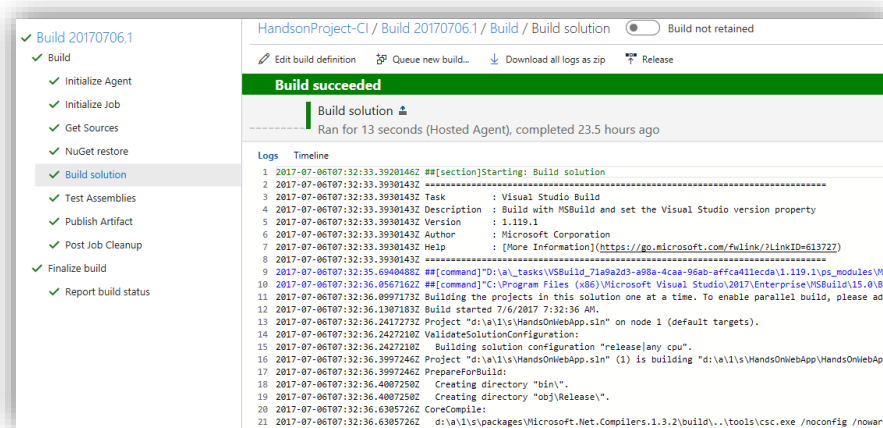
●NuGet restore

NuGet リポジトリからアプリのビルドに必要なパッケージを取得します。



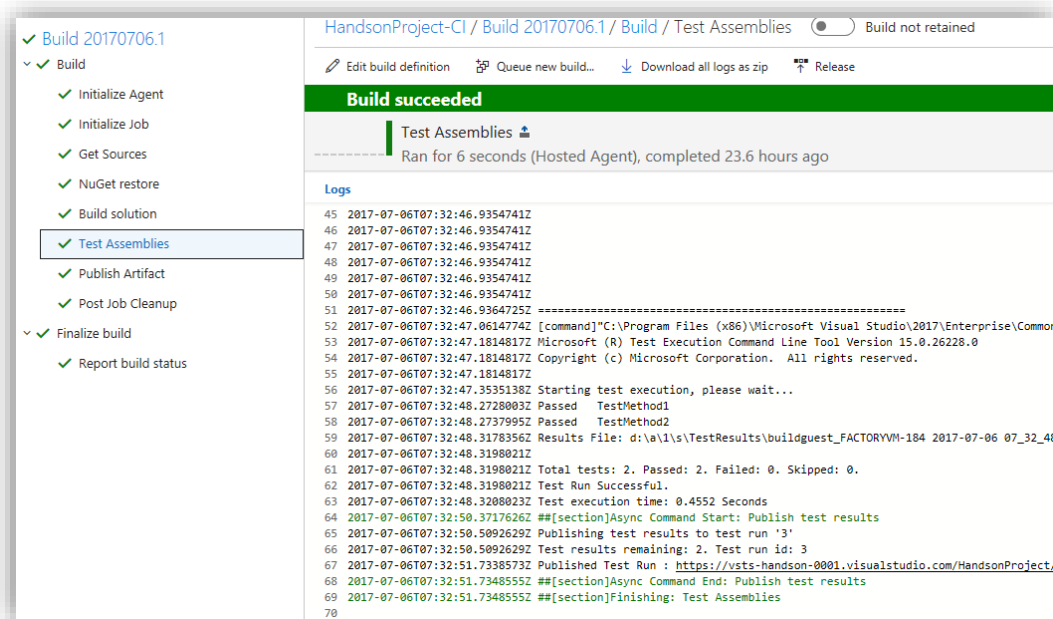
● Build solution

コードからアプリをビルド (コンパイル、Artifact の生成など) します。



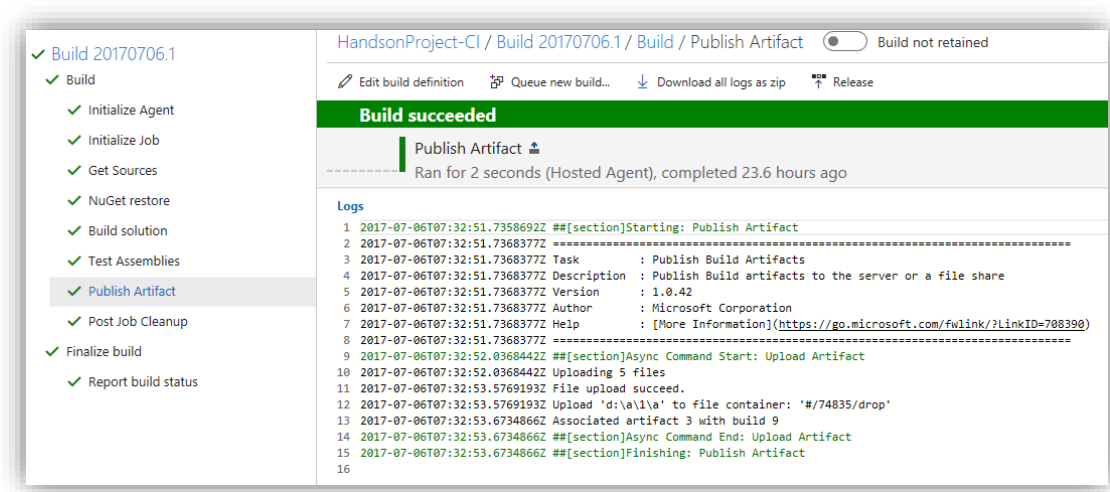
● Test Assemblies

Unit Test を実行します。



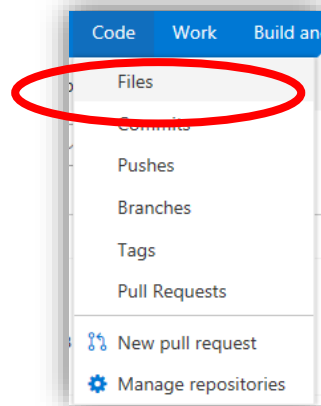
● Publish Artifact

作成された **Artifact** を取得できるようにします。

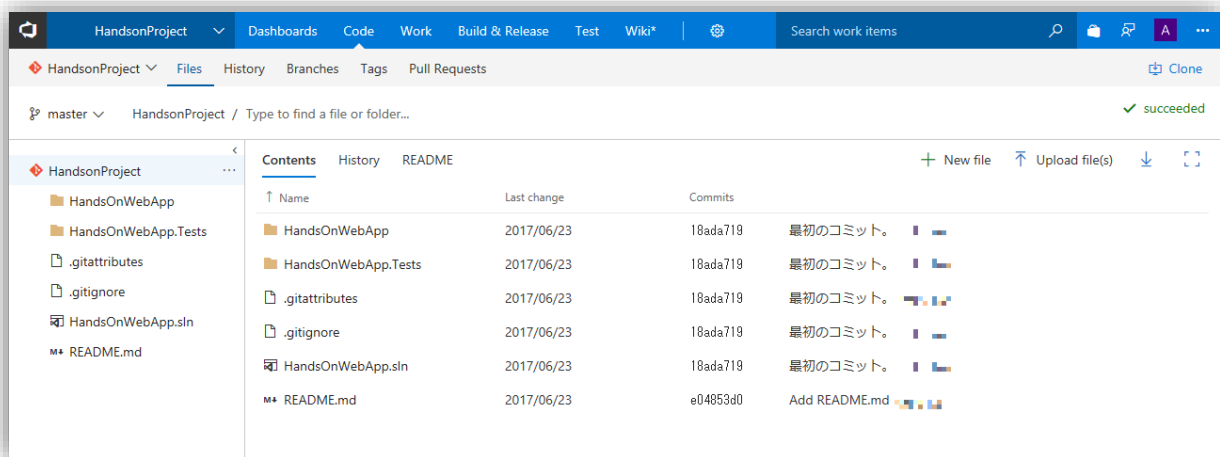


これで、ビルドと自動テストの実行が確認できました。以後は、**Visual Studio Code** でコードを編集して **push** すれば、その都度、ビルドパイプラインが実行されます。

20. 続いて、Git リポジトリを確認します。メニューバーの [Code] – [Files] をクリックします。

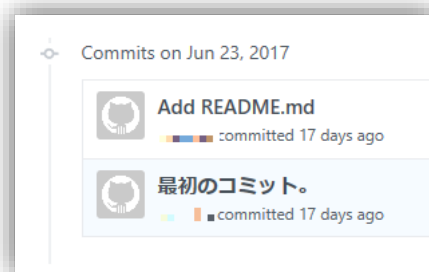
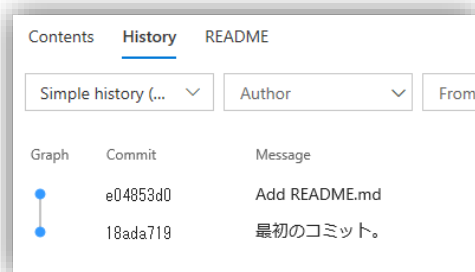


21. サンプルアプリのコードが push されていることを確認します。



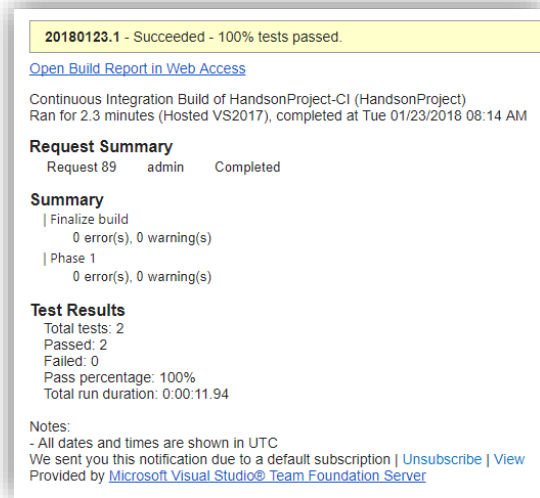
ワンポイント

「git clone」コマンドは、“リポジトリをそのままコピー” します。よって、History (履歴) もそのままコピーされるので、現時点では Github リポジトリと Visual Studio Team Service の Git リポジトリは全く同じ状態となります。

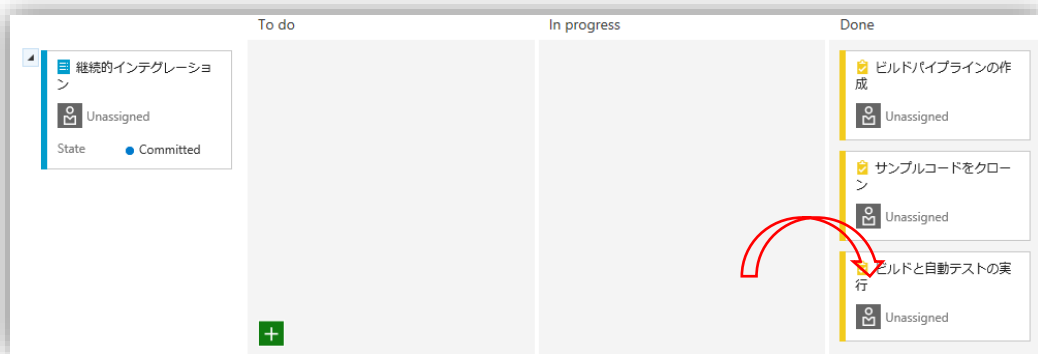


本自習書では、以後は Visual Studio Team Services の Git リポジトリを使用します。

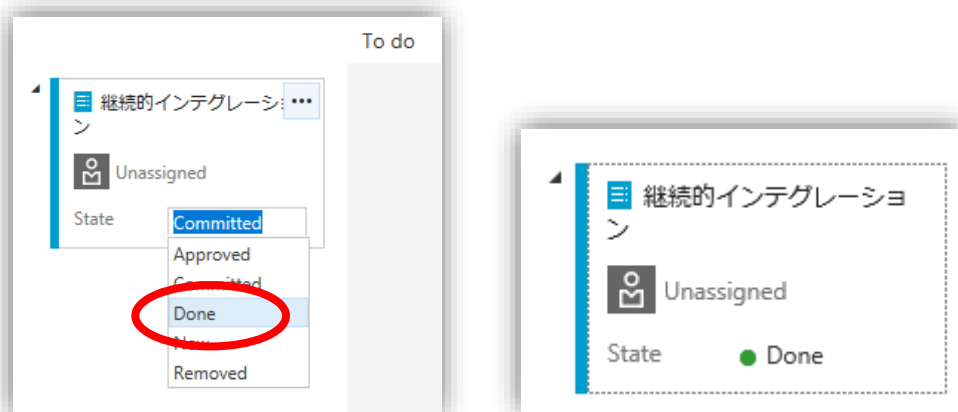
22. 続いて、Visual Studio Team Services にサインインしているユーザーのメールアドレス宛に、ビルド結果のメールが届いていることを確認します。



23. 最後に、Visual Studio Team Services の [Work] - [Backlogs] - [Sprint 1] を開きます。これでビルドと自動テストの実行は完了のため、Task [ビルドと自動テストの実行] を [In Progress] から [Done] にドラッグ & ドロップします。



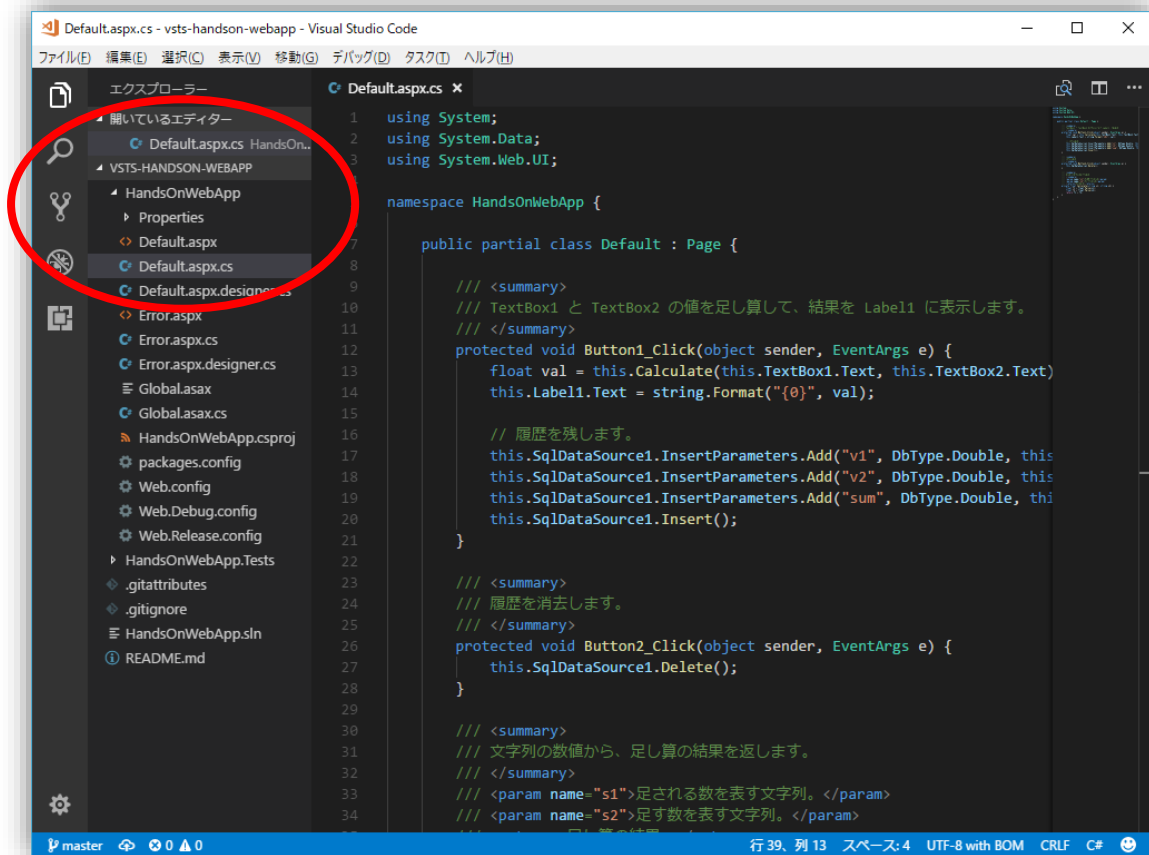
24. これで、[継続的インテグレーション] の Task はすべて完了したので、ステータスを [Committed] から [Done] に変更します。



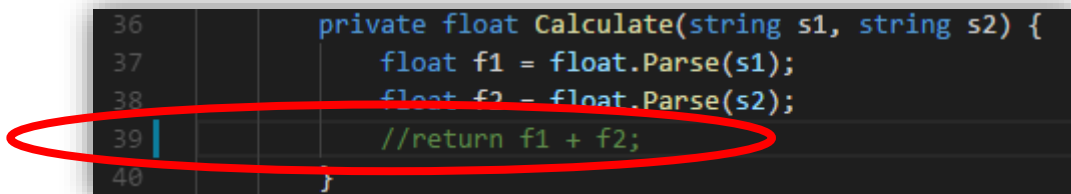
18. ビルドエラー時の挙動

次の手順では、Visual Studio Code を使用してサンプルアプリのコードを改変し、わざとビルドエラーを発生させます。このとき、ビルドパイプラインでどのような挙動を示すかを確認します。

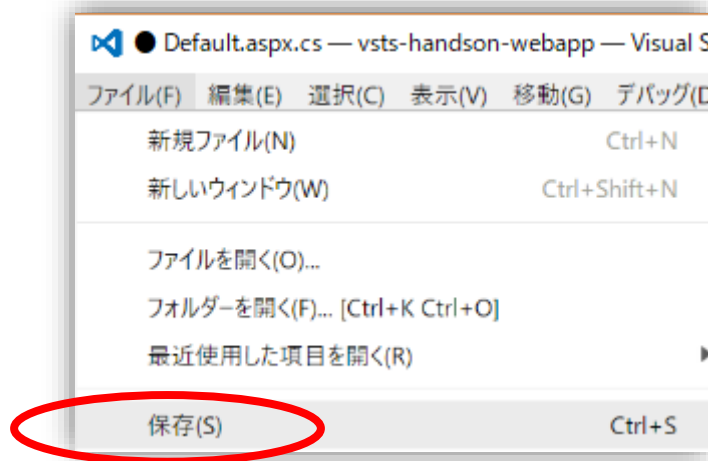
1. Visual Studio Code の [エクスプローラー] から、[VSTS-HANDSON-WEBAPP] – [HandsOnWebApp] – [Default.aspx.cs] をダブルクリックし、[Default.aspx.cs] を開きます。



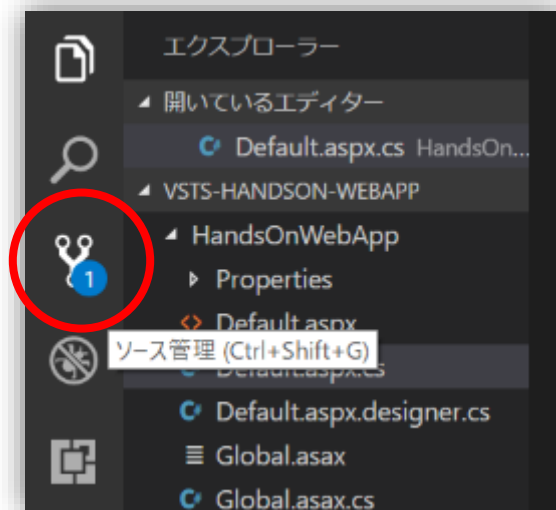
2. 39 行目の [return f1 + f2;] の前に「//」と入力し、この行をコメントアウトします。



3. メニューバーの [ファイル] - [保存] をクリックします。

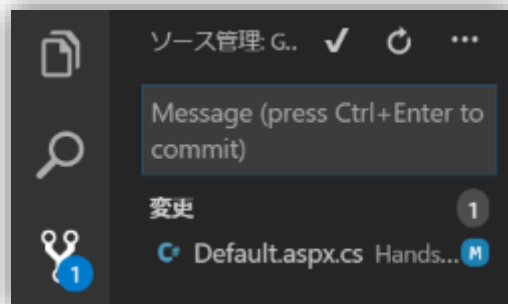


4. 画面左のツールバーから、[ソース管理] をクリックします。

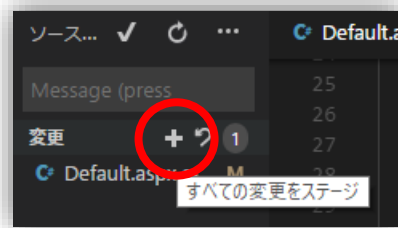


5. ソース管理の画面が表示されます。ここでは、変更があったファイルの一覧や、commit などのコマンド操作ができます。

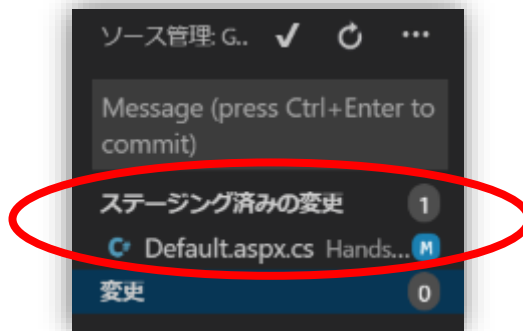
現状では、 [Default.aspx.cs] ファイルのみ変更があったので、このファイルが [変更] に表示されています。



6. [変更] にマウスカーソルを乗せ、[+] (すべての変更をステージ) をクリックします。



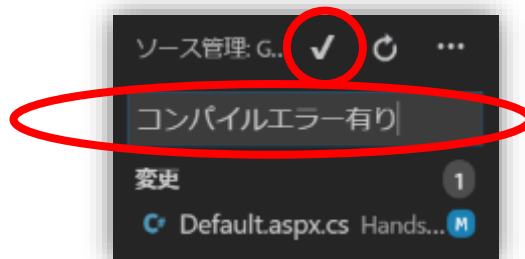
7. [Default.aspx.cs] ファイルが、[ステージング済みの変更] に移行します。



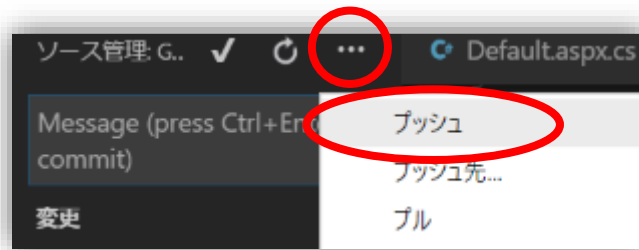
ワンポイント

Git では、ファイルが変更されてもすぐに `commit` できる状態にはなりません。まずは、ステージングエリアに追加 (`git add` コマンド) することで、次回の `commit` 対象になることを明示する必要があります。

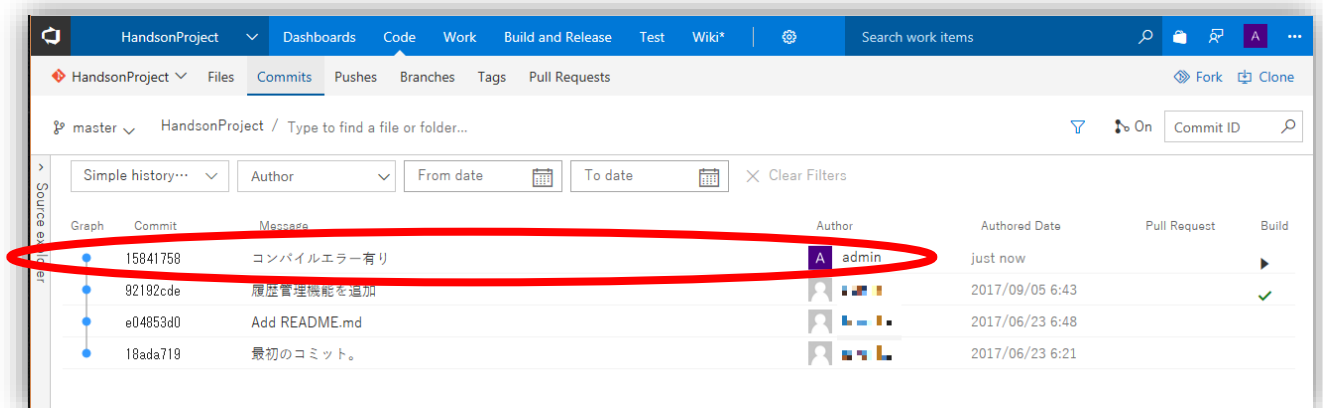
8. [Message] に「コンパイルエラー有り」と入力し、[✓] (Commit) をクリックします。



9. 続いて、[...]–[プッシュ] をクリックします。これで Visual Studio Team Services の Git リポジトリに push されます。

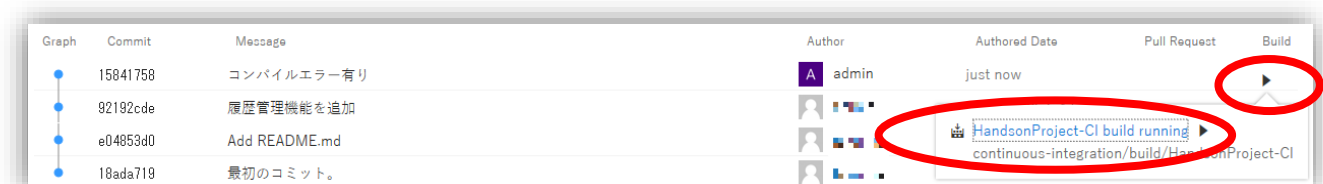


10. Visual Studio Team Services の [Code] 画面から [Commits] タブをクリックして履歴を表示します。[コンパイルエラー有り] のメッセージで commit した変更分が push されているのがわかります。

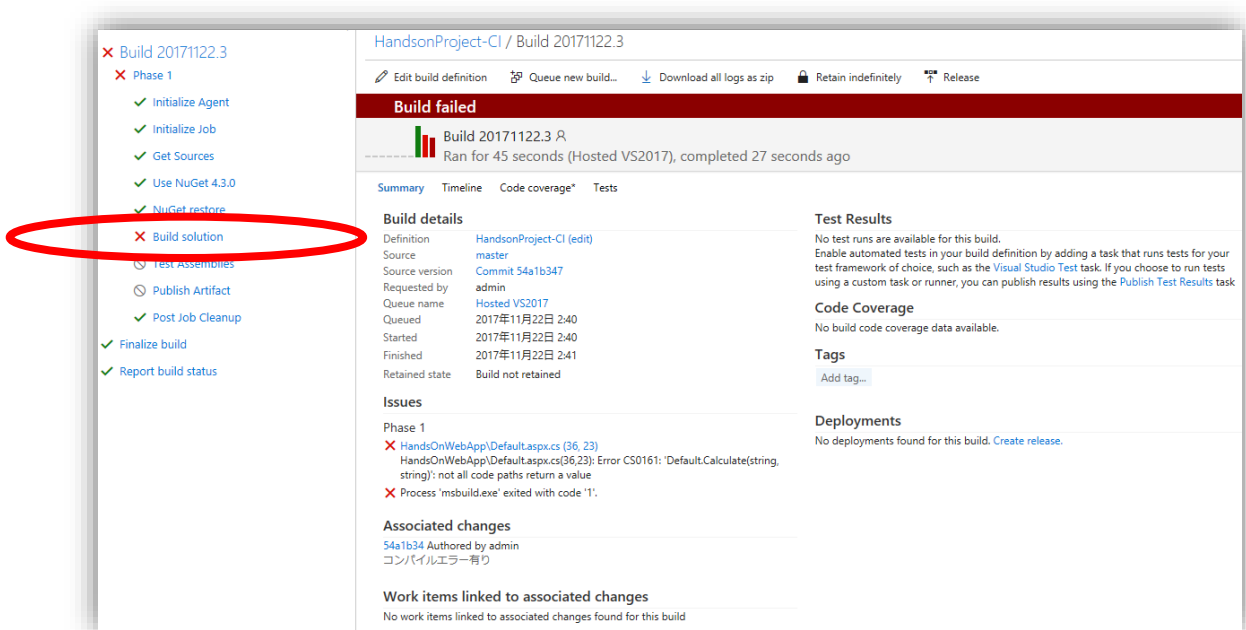


11. また、ビルドパイプラインを作成済みなので、[Build] から現在のステータスがわかります。

▶ は、ビルドが進行中であることを表します。この ▶ をクリックし、[HandsonProject-CI build running] をクリックします。



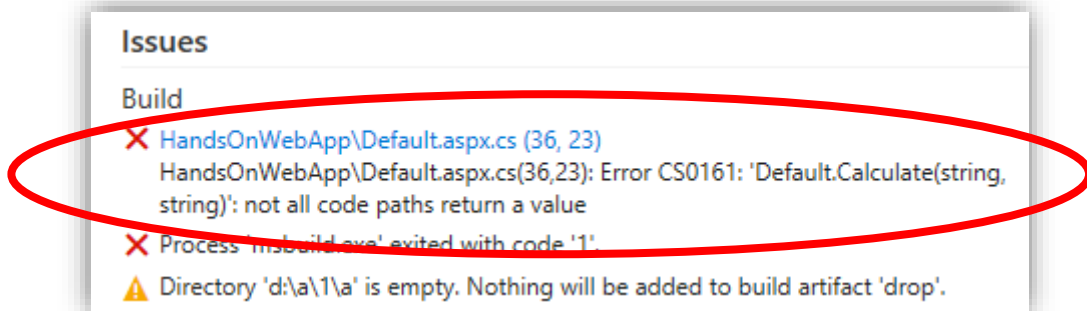
12. 今回の push で実行されたビルドパイプラインの Summary が表示されます。画面左のビルドプロセスの [Build solution] が × になっていることを確認します。



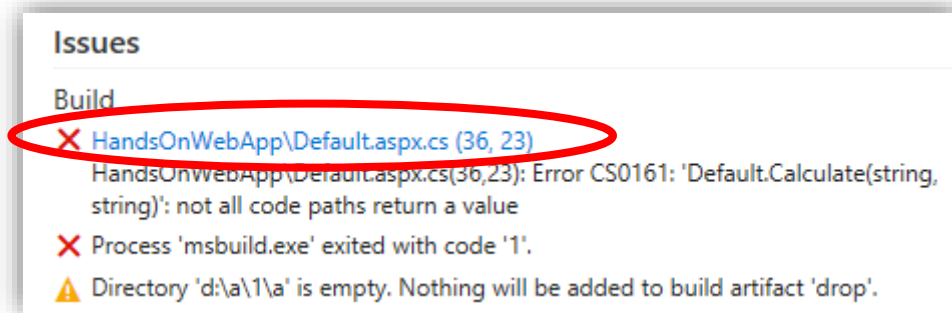
ワンポイント

[Build solution] でビルドが失敗したため、次のプロセスの [Test Assemblies] は実行されません。

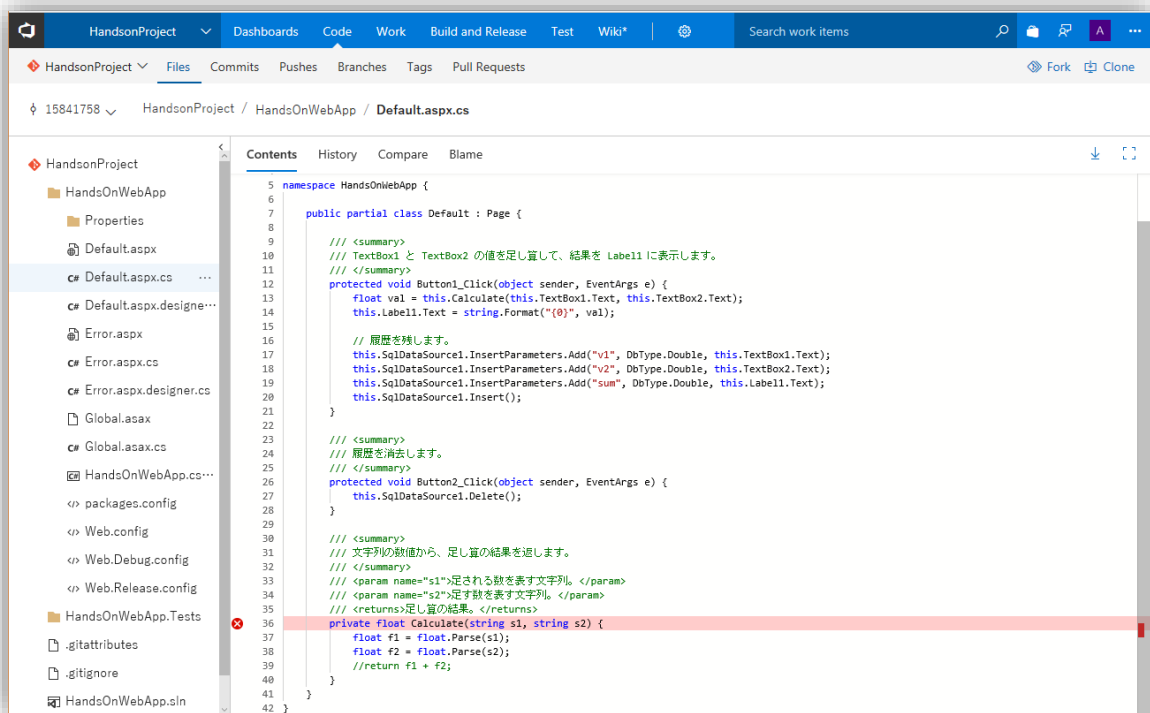
13. [Issues] では、どのファイルのどの部分に問題があったかが表示されます。
以下のメッセージから、Default.aspx.cs の関数 Calculate が戻り値を返していないために、ビルドエラーが発生したことがわかります。



14. 問題があったファイルがリンクになっているので、これをクリックします。



15. Web ブラウザーの新しいタブで、Git リポジトリの画面が表示され、問題のあるコードがハイライトされます。コードのどの部分に問題があったかが明確になるため、修正時に役立ちます。



16. ビルド失敗を通知するメールが届いていることを確認します。

20180123.2 - Failed

[Open Build Report in Web Access](#)

Continuous Integration Build of HandsonProject-CI (HandsonProject)
Ran for 1.5 minutes (Hosted VS2017), completed at Tue 01/23/2018 09:26 AM

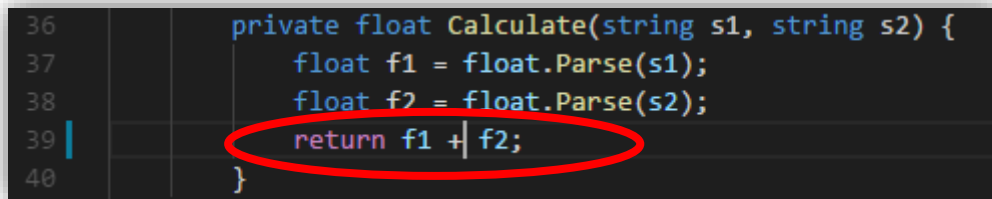
Request Summary
Request 90 admin Completed

Summary
| Finalize build
 0 error(s), 0 warning(s)
| Phase 1
 2 error(s), 0 warning(s)
 Phase 1 - 2 error(s), 0 warning(s)
 HandsOnWebApp!Default.aspx.cs(36):HandsOnWebApp!Default.aspx.cs(36,23): Error CS0161: 'Default.Calculate(string, string)': not all code paths return a value
 Process 'msbuild.exe' exited with code '1'.

19. テスト失敗とカンバンボードに Bug を追加

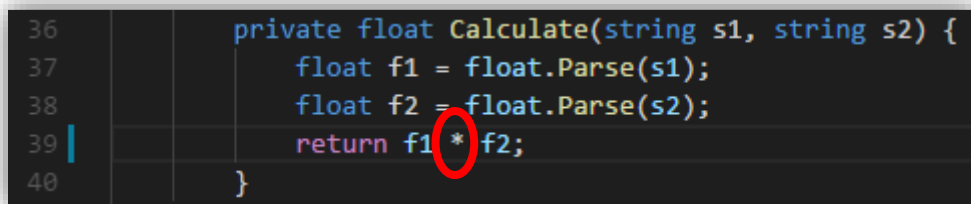
次の手順では、Visual Studio Code を使用してサンプルアプリのコードを改変し、わざとテストを失敗させます。このとき、ビルドパイプラインでどのような挙動を示すかを確認し、発生したエラーを修正するため、カンバンボードに Bug を追加します。

1. まずは、先ほどのビルドエラーのコードを元に戻します。Visual Studio Code で Default.aspx.cs を開き、39 行目のコメントアウトを削除します。これでビルドエラーが修正されます。



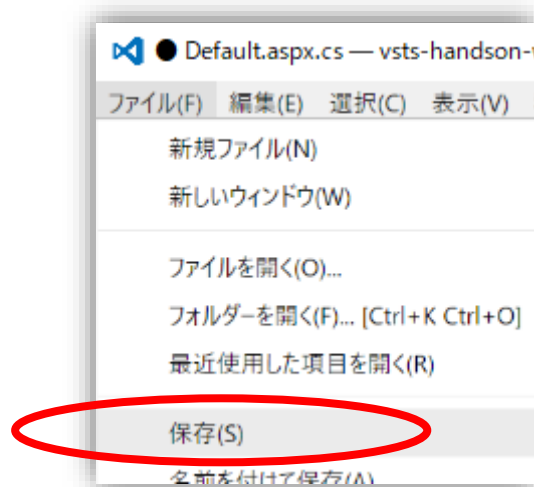
```
36 private float Calculate(string s1, string s2) {  
37     float f1 = float.Parse(s1);  
38     float f2 = float.Parse(s2);  
39     return f1 + f2;  
40 }
```

2. 引き続き、同じ 39 行目の [+] を、「*」に変更します。**Calculate** 関数は足し算の結果を返すはずなのに、間違って掛け算の結果を返すように実装してしまったことを想定しています。



```
36 private float Calculate(string s1, string s2) {  
37     float f1 = float.Parse(s1);  
38     float f2 = float.Parse(s2);  
39     return f1 * f2;  
40 }
```

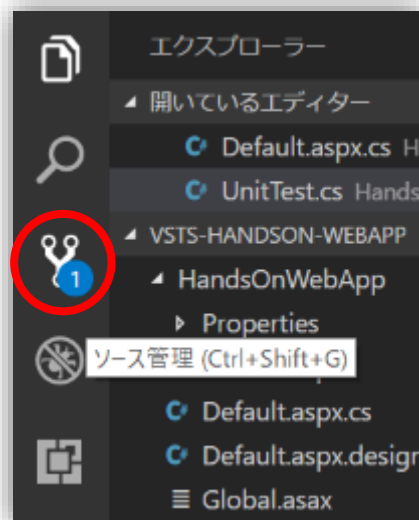
3. メニューバーの [ファイル] - [保存] をクリックして、Default.aspx.cs の変更を保存します。



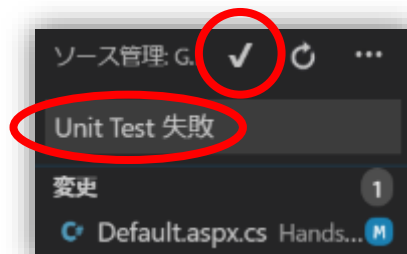
4. [エクスプローラー] から、[VSTS-HANDSON-WEBAPP]- [HandsOnWebApp.Tests] – [UnitTest.cs] をダブルクリックして開きます。これは Unit Test が記述されたファイルです。
9 ～ 14 行目のテスト TestMethod1 を確認します。これは、Calculate 関数が正しく足し算の結果を返すかをテストしますが、先ほど足し算を掛け算に変更したので、このテストは失敗します。



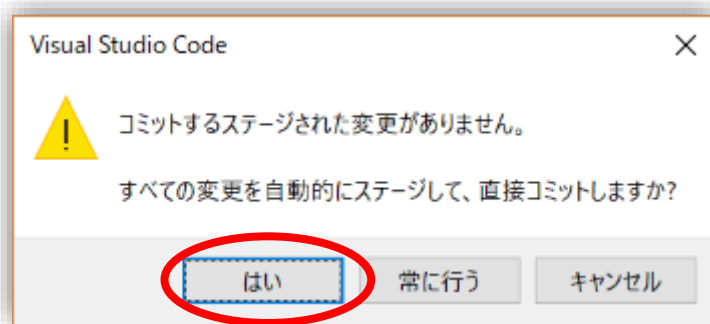
5. 画面左のツールバーから [ソース管理] をクリックします。



6. [ソース管理] の [Message] に「Unit Test 失敗」と入力し、[✓] (Commit) をクリックします。



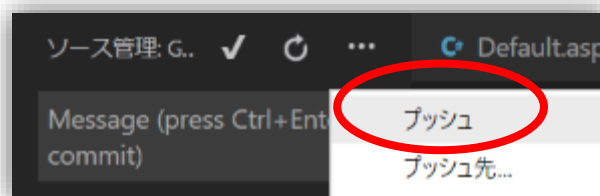
7. 以下のメッセージが表示されるので、[はい] をクリックします。



ワンポイント

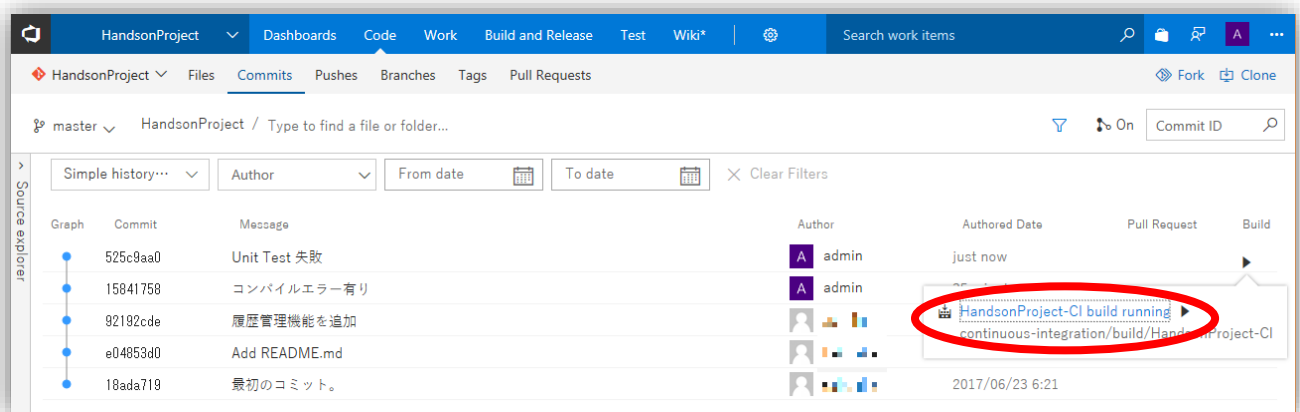
ステージングエリアへの追加を行わずに `commit` をクリックすると、すべての変更をステージングエリアに自動で追加した後で `commit` させる (ステージング作業の省略) ことができます。

8. [...] – [プッシュ] をクリックして、リモートリポジトリに `push` します。

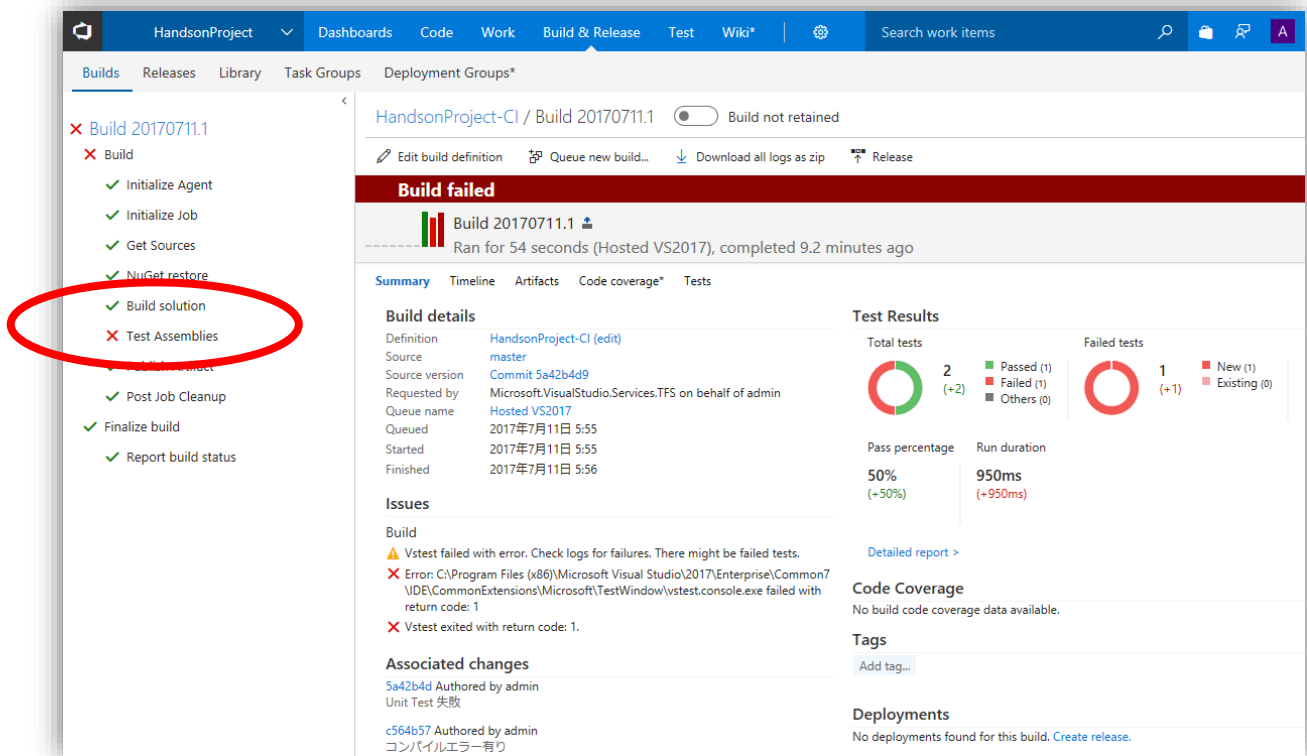


9. Visual Studio Team Services の [Code] 画面から [Commits] タブをクリックして履歴を表示します。
[Unit Test 失敗] のメッセージで `commit` した変更分が `push` されます。

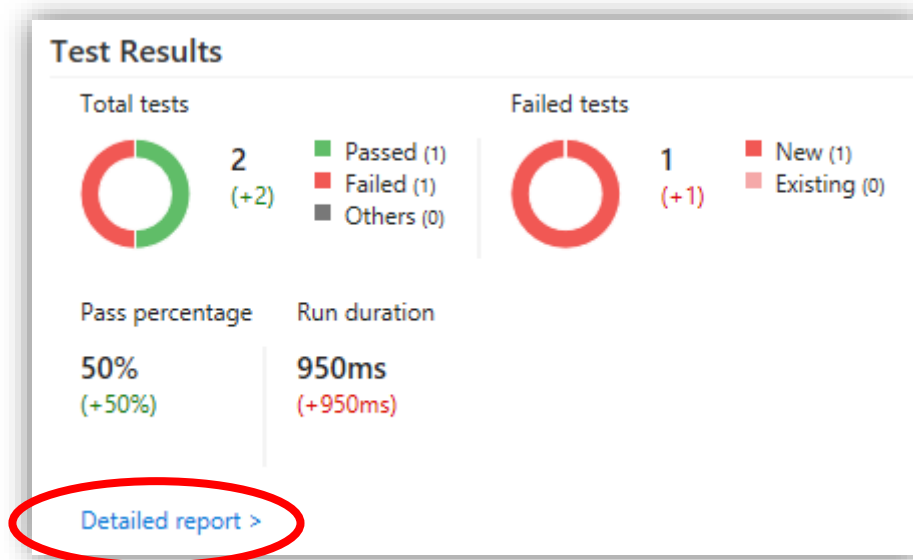
この [Build] の  をクリックし、[HandsonProject-CI build running] をクリックします。



10. 今回の push で実行されたビルドパイプラインの Summary が表示されます。画面左のビルドプロセスの [Build solution] が ✓ になり、ビルドは成功していることを確認します。また、[Test Assemblies] が ✗ になり、エラーになっていることを確認します。



11. [Test Results] では、テストステータスを確認できます。全部で2つあるテストのうち、1つが成功し1つが失敗していることがわかります。この [Detailed report] をクリックします。



12. テスト結果の詳細が表示されます。

[TestMethod1] をクリックすると、画面右下にエラーメッセージが表示されます。このメッセージから、要求される値は **3** なのに、実際には **2** が返ってきたことがわかります。

HandsonProject-CI / Build 20170711.1 Build not retained

Edit build definition Queue new build... Download all logs as zip Release

Build failed

Build 20170711.1
Ran for 54 seconds (Hosted VS2017), completed 9.2 minutes ago

Summary Timeline Artifacts Code coverage* Tests

Total tests: 2 (+2) (1 Passed, 1 Failed, 0 Others)
Failed tests: 1 (+1) (1 New, 0 Existing)
Pass percentage: 50% (+50%)
Run duration: 950ms (+950ms)

Test failures: 1
Test duration: 950ms

Group by Test run Outcome Failed

Test	Failing since	Failing build	Duration
1/2 Passed - VSTest Test Run release an...			0:00:00.196
✖ TestMethod1 New	17 minutes...	Current build	0:00:00.183

TestMethod1
Failed on FACTORYVM-172
Duration 0:00:00.183, 20 minutes ago

Error message
Assert.AreEqual failed. Expected:<3>. Actual:<2>.

Stack trace
at HandsOnWebApp.Test.UnitTest.TestMethod1() in d:\a\...

Attachments (0) Bugs (0)
No attachments No linked bugs

Requirements (0)

13. テスト失敗の通知メールが送信されていることを確認します。

20180123.3 - Failed - 50% tests passed.

[Open Build Report in Web Access](#)

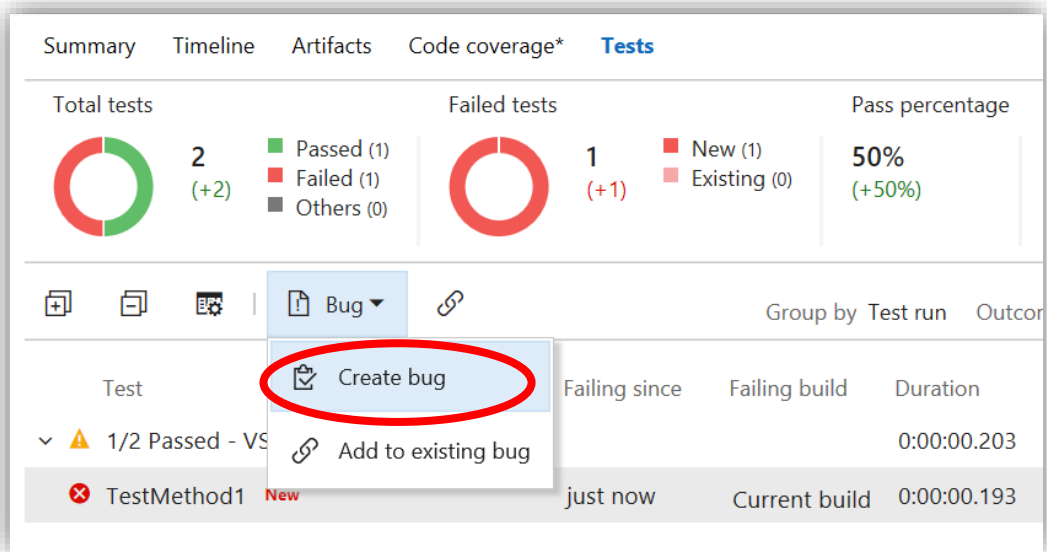
Continuous Integration Build of HandsonProject-CI (HandsonProject)
Ran for 2.4 minutes (Hosted VS2017), completed at Tue 01/23/2018 09:33 AM

Request Summary
Request 91 admin Completed

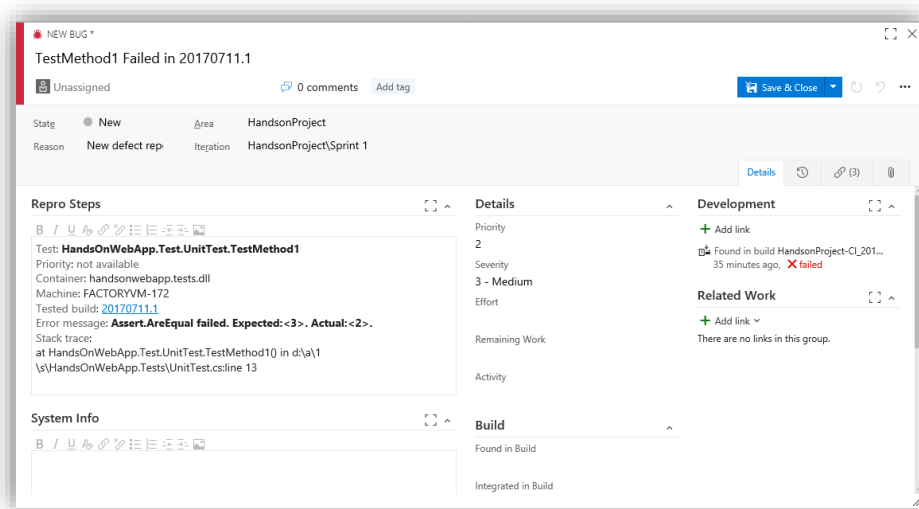
Summary
| Finalize build
0 error(s), 0 warning(s)
| Phase 1
2 error(s), 1 warning(s)
Phase 1 - 2 error(s), 1 warning(s)
Error: C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\Common7\IDE\CommonExtensions\Microsoft\TestWindow\vstest.console.exe failed with return code: 1
VsTest task failed.

Test Results
Total tests: 2
Passed: 1
Failed: 1
Pass percentage: 50%
Total run duration: 0:00:11.377

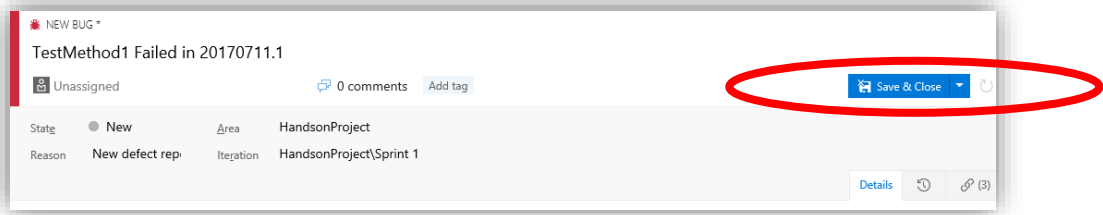
14. このテスト結果を、カンバンボードに **Bug** として登録します。**[TestMethod1]** を選択した状態で、ツールバーから **[Bug] - [Create bug]** をクリックします。



15. 新しい **Bug** を作成するためのダイアログが表示されます。
[Repro Steps] には、テストメソッドやエラーメッセージなどの情報が追加されます。
[Development] には、このエラーが発生したビルドパイプラインの **Summary** へのリンクが追加されます。



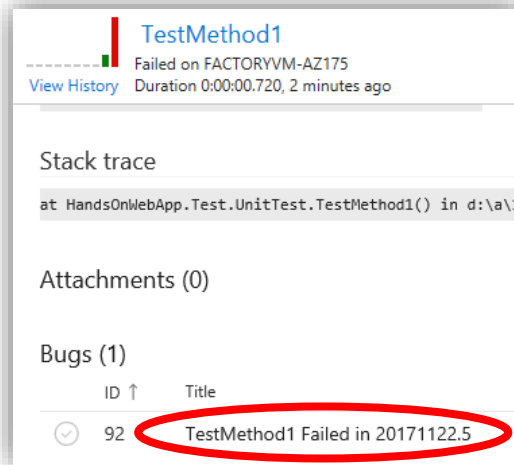
16. 特に何も変更せず、[Save & Close] をクリックします。



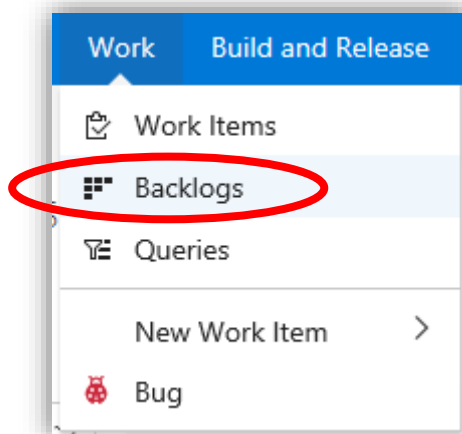
ワンポイント

実際には、Bug を修正する担当者をアサインする必要がありますが、本自習書では省略します。

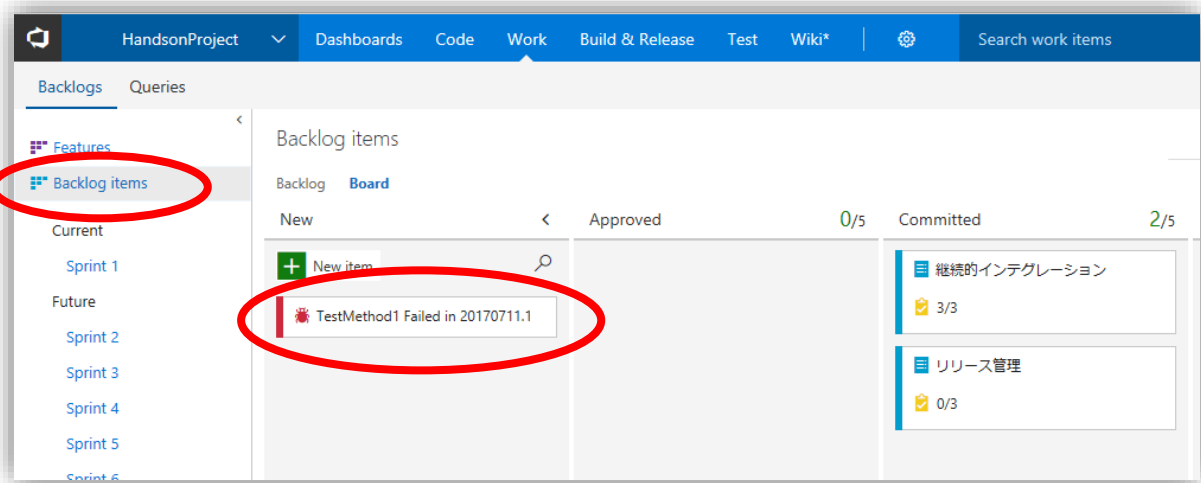
17. [TestMethod1] の [Bugs] に、作成した Bug が紐づけられます。Bug に対して ID が自動採番されますが、この ID を控えておきます。以下の場合には 92 です。



18. この Bug を修正するための Task を追加するため、[Work] - [Backlogs] をクリックします。



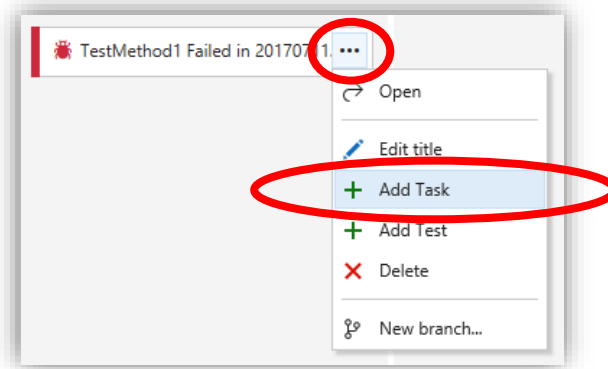
19. 画面左から、[Backlog items] をクリックします。カンバンボードの [New] に先ほど作成した Bug が表示されていることを確認します。



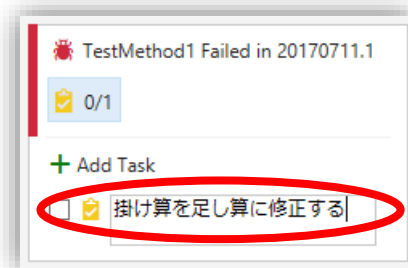
ワンポイント

Bug も Product Backlog と同じように、配下に Task を作成して修正作業を管理します。

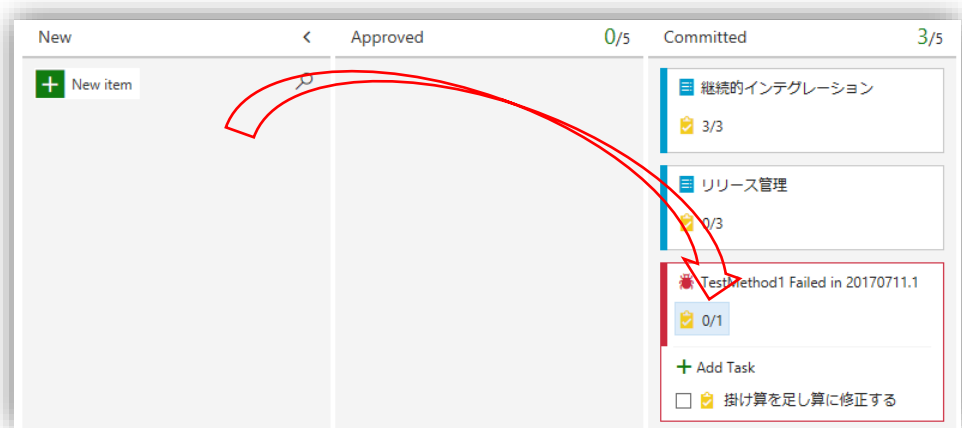
20. この Bug の [...] – [Add Task] をクリックします。



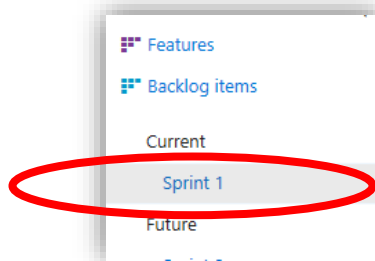
21. Task のカンバンが作成されるので、「掛け算を足し算に修正する」と入力します。



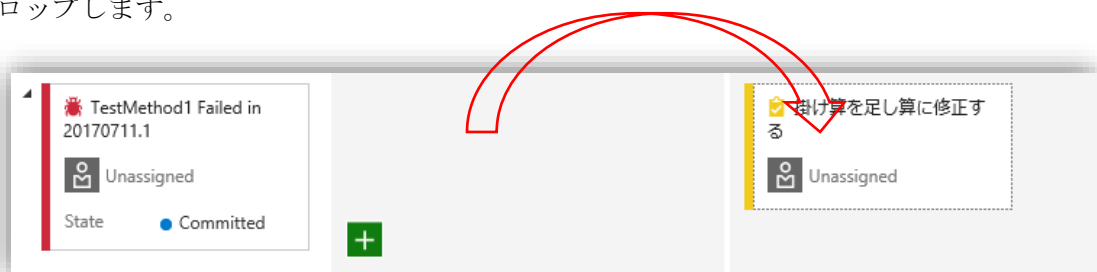
22. Bug を [New] から [Committed] にドラッグ & ドロップします。



23. 画面左から [Spring 1] をクリックします。



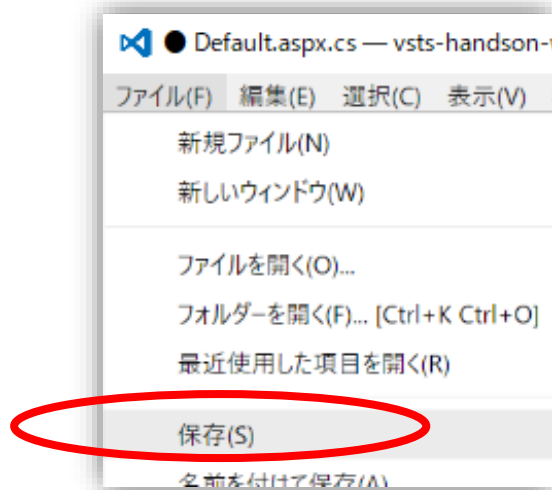
24. 画面下方にある Task [掛け算を足し算に修正する] を [To do] から [In progress] にドラッグ & ドロップします。



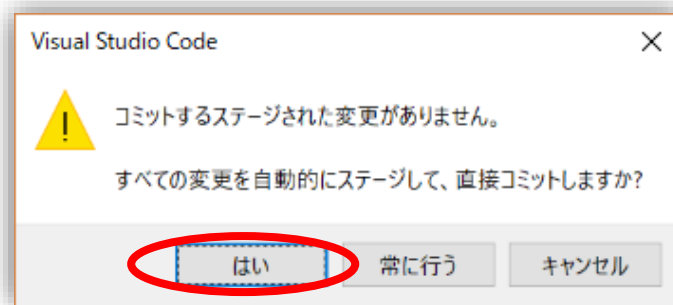
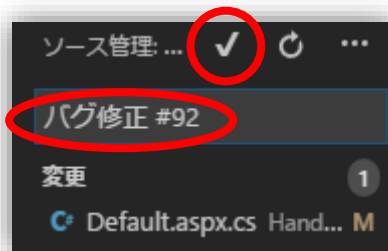
25. Visual Studio Code で開いている [Default.aspx.cs] の 39 行目を、掛け算「*」から足し算「+」に修正します。

```
36 private float Calculate(string s1, string s2) {  
37     float f1 = float.Parse(s1);  
38     float f2 = float.Parse(s2);  
39     return f1 + f2;  
40 }
```

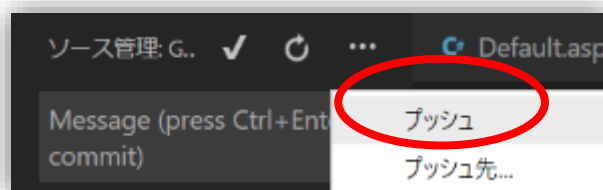
26. メニューバーの [ファイル] - [保存] をクリックして、Default.aspx.cs の変更を保存します。



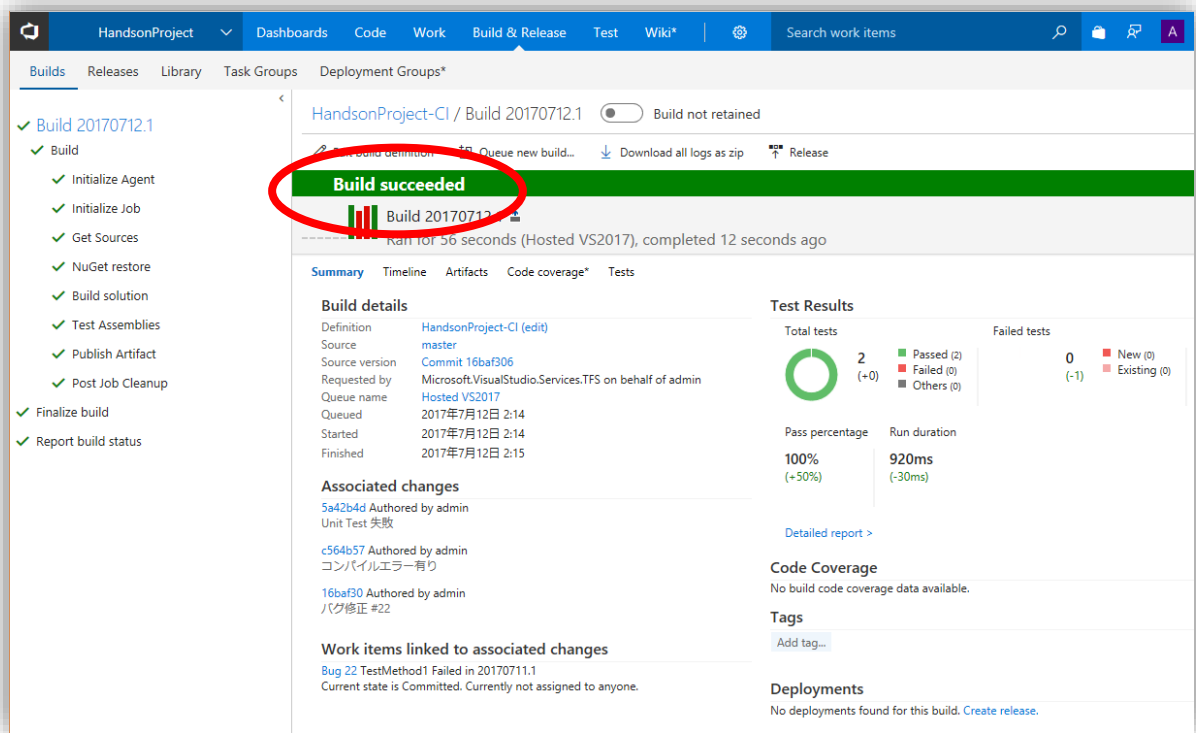
27. [ソース管理] の [Message] に「バグ修正 #<ID>」と入力し、[✓] (Commit) をクリックします。
ID は **Bug** 作成時に自動採番された番号で、先ほど控えていた値です。
メッセージボックスが表示されたら [はい] をクリックします。



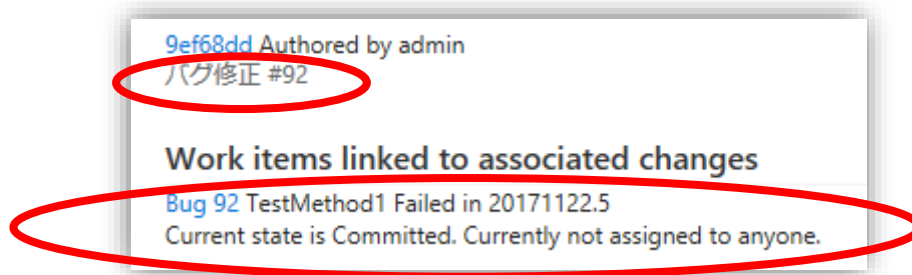
28. [...] - [プッシュ] をクリックして、リモートリポジトリに push します。



29. Visual Studio Team Services で、先ほどの push で実行されたビルドパイプラインの結果の Summary を表示します。バグが修正され、ビルド結果が **succeeded** になることを確認します。



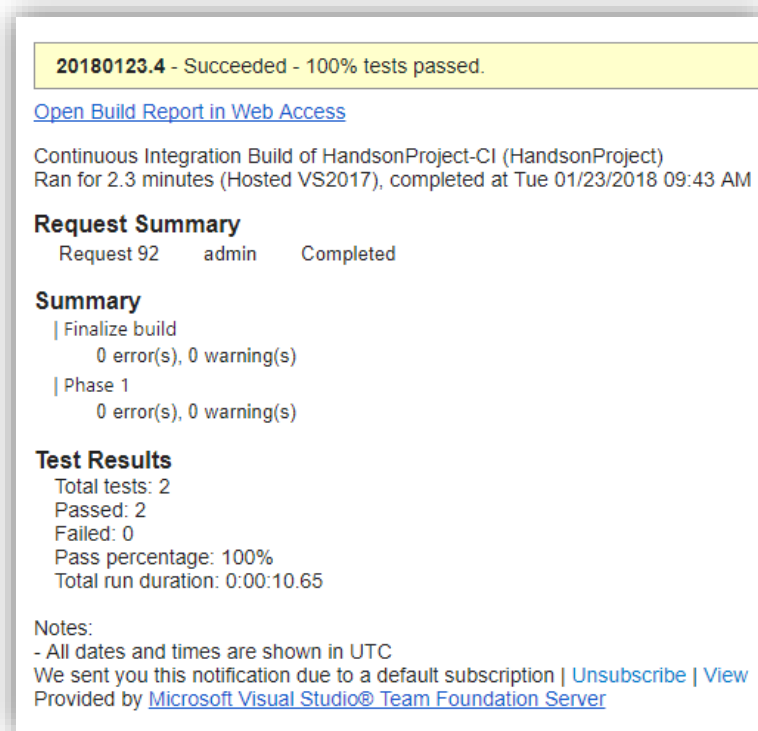
30. [Work items linked to associated changes] には、カンバンボードに作成した Bug へのリンクが貼り付けられます。Commit Message に「#<ID>」を追加すると、このビルドと Bug に関連性を持たせることができます。



ワンポイント

Bug だけではなく、他の Work item にも関連性を持たせることができます。なお、現状ではリンクが貼られるだけで、Work item のステータス (To do, In progress など) を Commit Message を利用して変更することはできません。

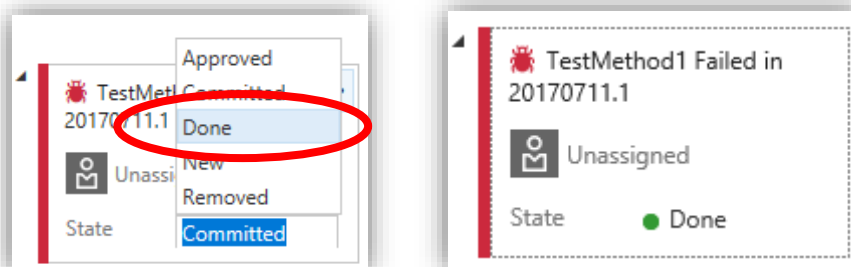
31. ビルド成功の通知メールが送信されていることを確認します。



32. Bug の修正が完了したので、カンバンボードの Task [掛け算を足し算に修正する] を [In progress] から [Done] にドラッグ & ドロップします。



33. これで、Bug の Task はすべて完了したので、ステータスを [Committed] から [Done] に変更します。



STEP 4. リリース管理

この STEP では、Azure App Service Web App に、ビルドしたアプリをデプロイするためのリリース管理について説明します。

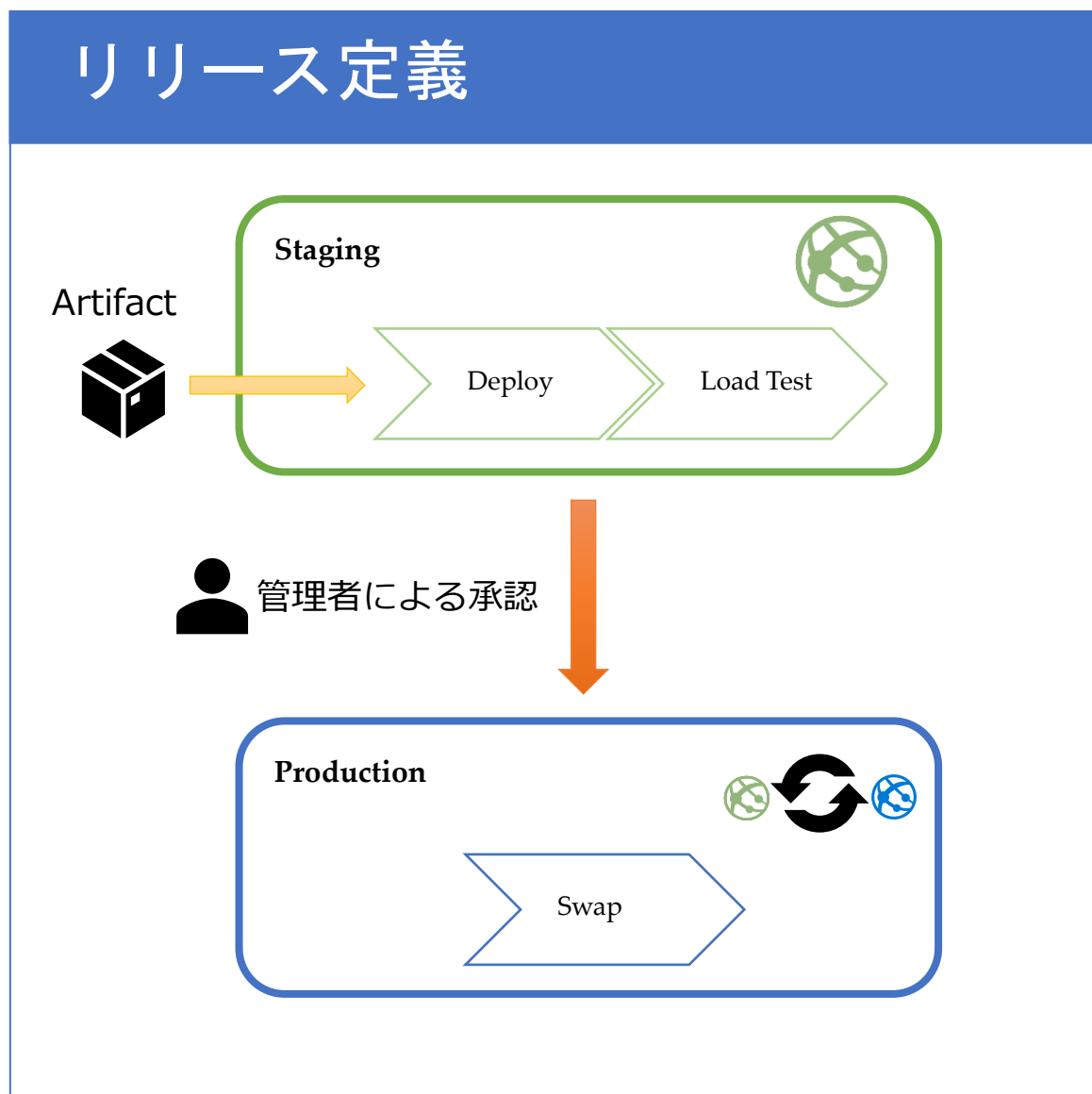
この STEP では、次のことを学習します。

- ✓ リリース定義の作成
- ✓ ステージング環境へのデプロイとロードテスト
- ✓ リリース環境へのスワッピング

20. リリース定義とは

リリース定義とは、ビルドしたアプリを Azure などの運用環境へデプロイするための定義です。定義内には複数の環境を追加することができ、環境ごとにデプロイプロセスを組むことができます。

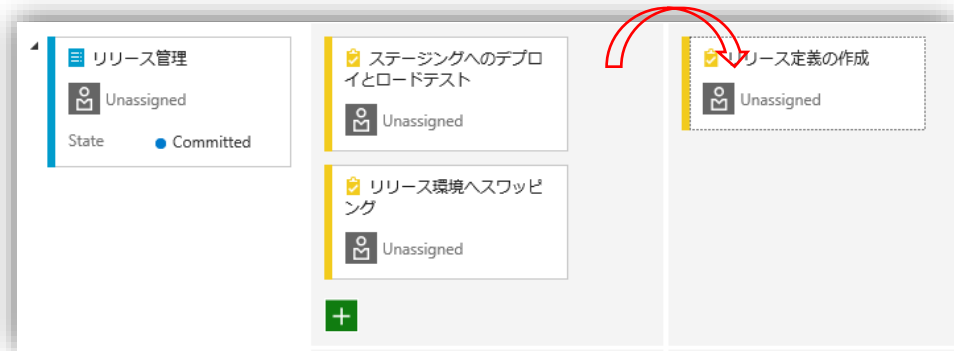
本自習書では、以下のように [Staging] と [Production] という 2 つの環境を作成し、完了したビルドから、まずは [Staging] 環境へのデプロイとロードテストを行います。続いて、管理者の承認を経て [Production] にスワッピング (交換) するリリース定義を作成します。



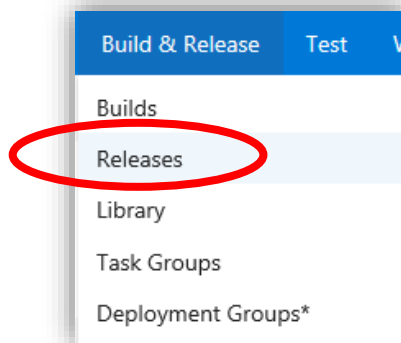
21. リリース定義の作成

次の手順では、リリース定義を作成します。

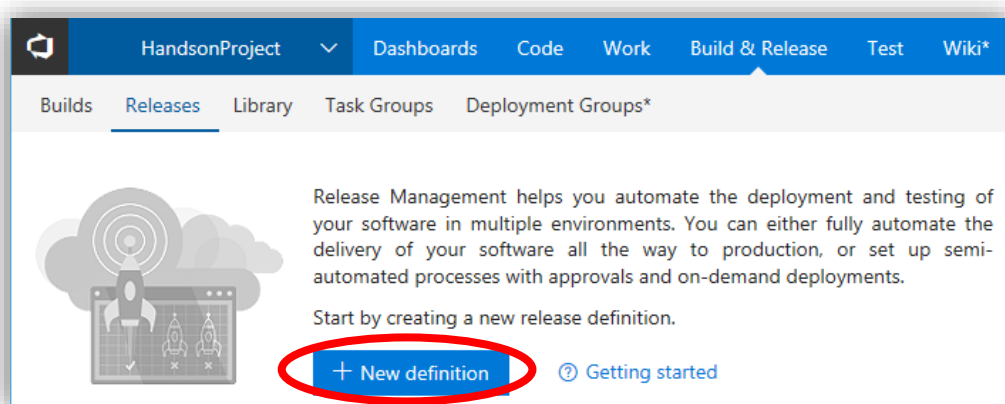
1. [Work] - [Backlogs] - [Sprint 1] を開きます。これから、リリース定義の作成を行うため、Product Backlog [リリース管理] の Task [リリース定義の作成] を [To do] から [In Progress] にドラッグ & ドロップします。



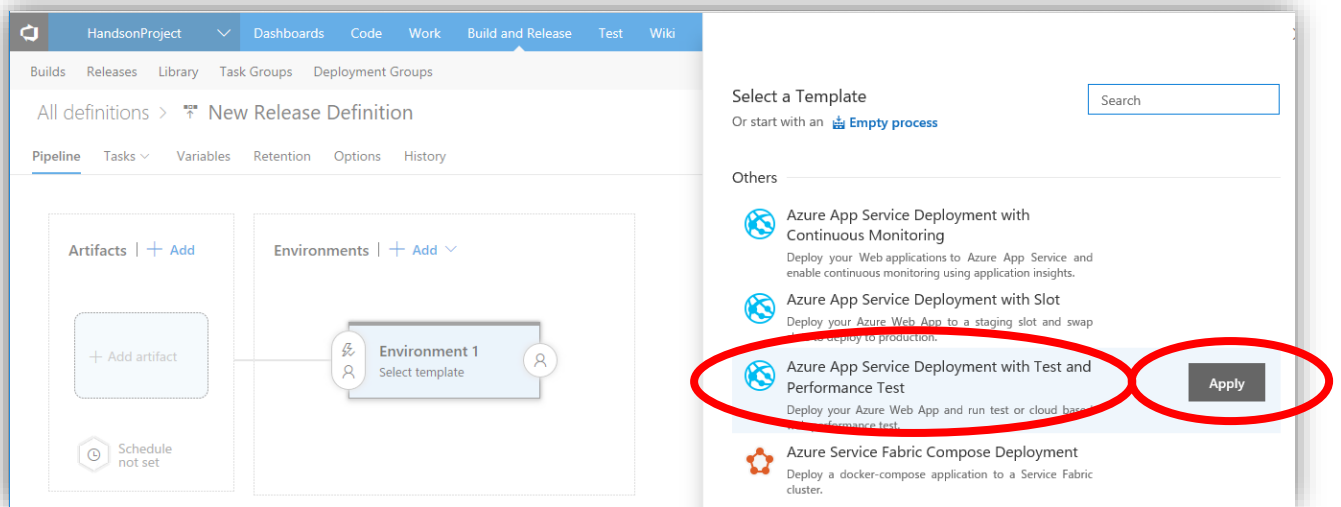
2. メニューバーから、[Build & Release] – [Releases] をクリックします。



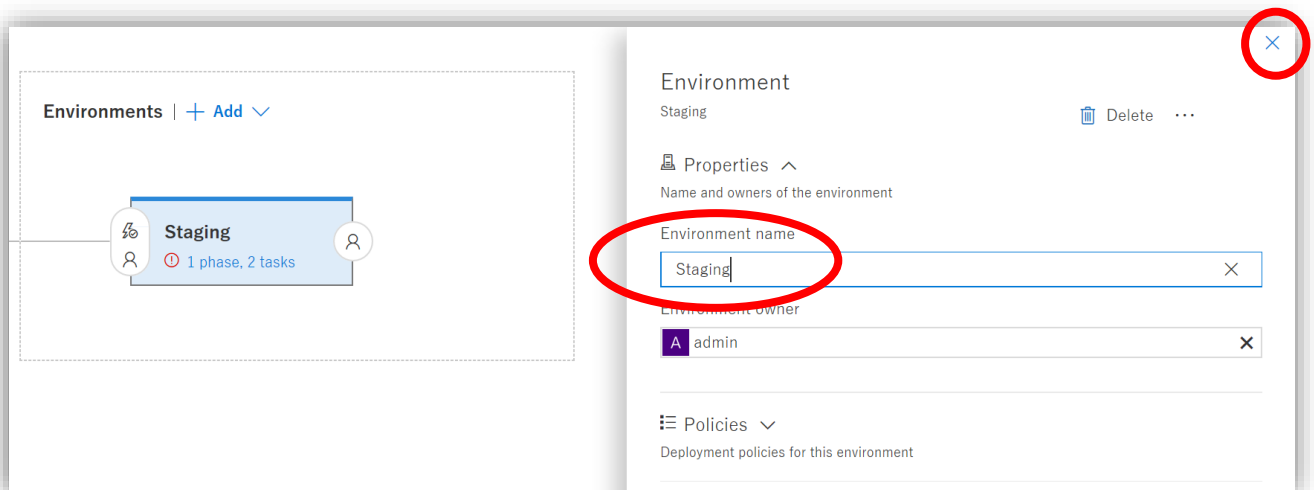
3. [New definition] をクリックします。



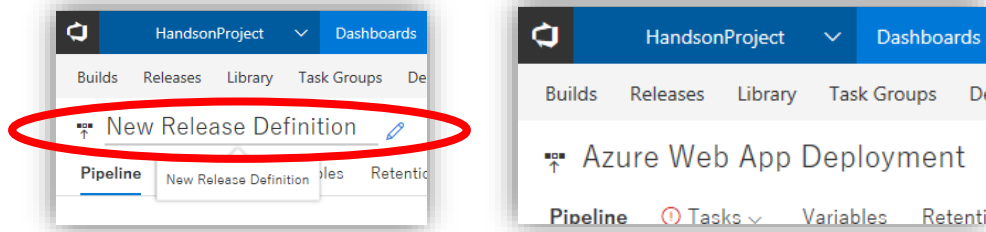
4. リリース定義の作成画面が表示されるので、画面右側の [Select a Template] から、[Azure App Service Deployment with Test and Performance Test] を選択し、[Apply] をクリックします。



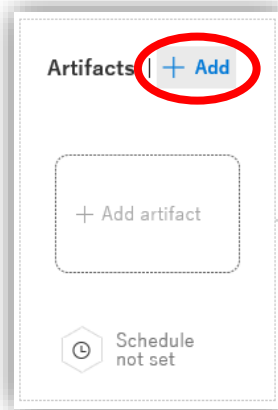
5. 環境の設定画面が表示されるので、[Environment name] に「Staging」と入力し、[×] をクリックします。



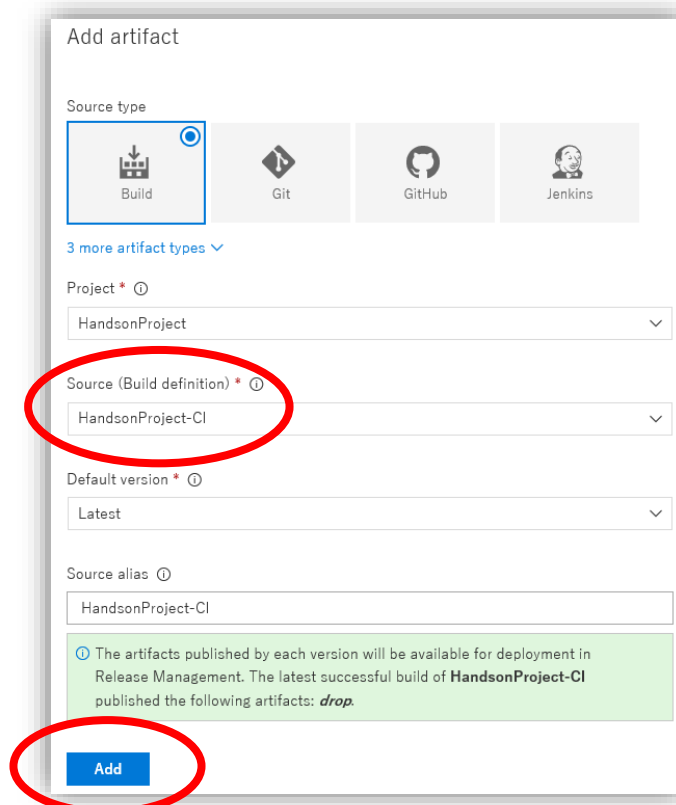
6. 次に、このリリース定義の名前を変更します。画面左上の [New Release Definition] をクリックすると定義の名前を変更できる状態になるので、「Azure Web App Deployment」と入力します。



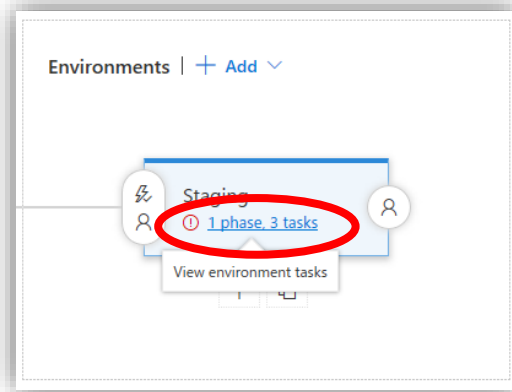
7. 次に、リリースに使用する Artifact を追加します。[Artifacts] の [+Add] をクリックします。



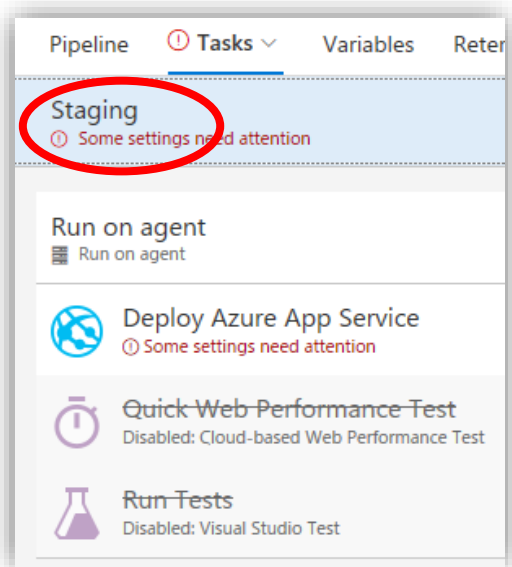
8. Artifact 追加画面が表示されるので、[Source (Build definition)] に先ほど作成したビルドパイプラインの [HandsonProject-CI] を選択し、[Add] をクリックします。



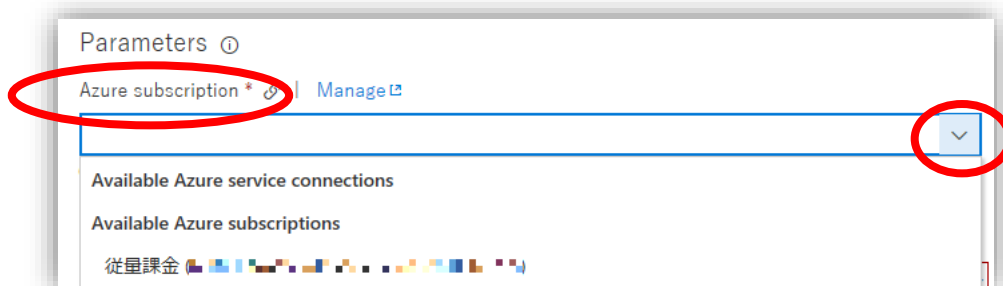
9. 次に、**Staging** 環境で実行するタスクの設定をします。**[Staging]** の **[1 phase, 3 tasks]** をクリックします。



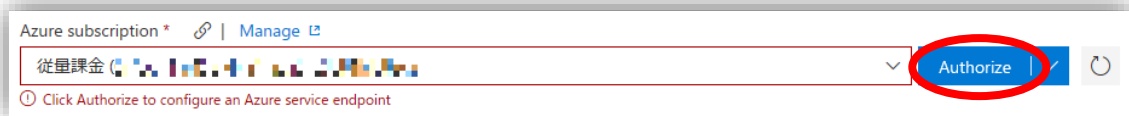
10. タスク一覧が表示されます。**[Staging]** が選択されていることを確認します。



11. **Azure** サブスクリプションの選択画面が表示されているので、**[Azure subscription]** のリストボックス右端の **[v]** をクリックしてサブスクリプションの一覧を表示し、**Web App** を作成したサブスクリプションを選択します。



12. [Authorize] をクリックします。



13. 認証のポップアップウィンドウが表示されるので、Visual Studio Team Services にサインイン中のアカウントと同じ Microsoft アカウントでサインインします。

※Microsoft アカウントの種類によっては、ポップアップが表示されずに認証が通る場合があります。

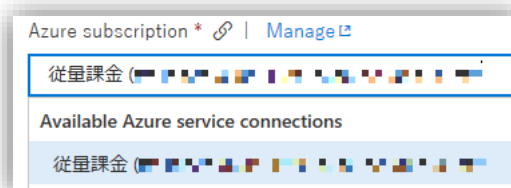


ワンポイント

Web ブラウザーの設定でポップアップがブロックされている場合、以下のようなエラーメッセージが表示されます。Web ブラウザーのポップアップブロックを解除し、再度 [Authorize] をクリックしてください。



14. 認証完了後、もう一度 [Azure subscription] のリストボックスをクリックします。選択したサブスクリプションが [Available Azure service connections] に表示されていることを確認します。

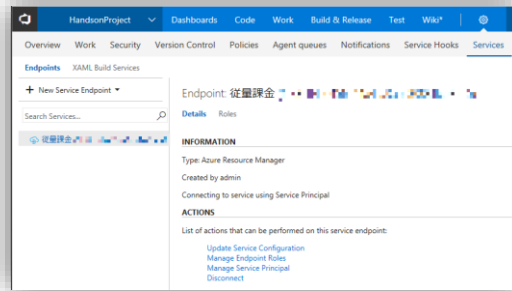


ワンポイント

[Manage] をクリックすると、サービスエンドポイントの管理画面が表示されます。

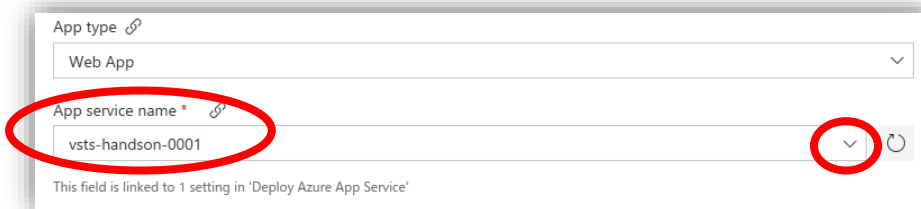


ここでは、認証済みの接続 (エンドポイント) を参照、削除することができます。また、新しいエンドポイントの作成も行うことができます。



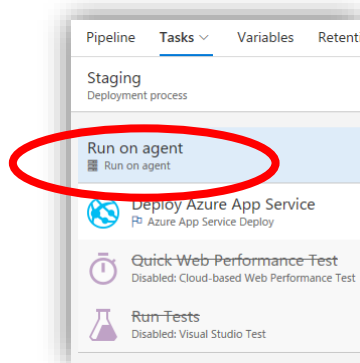
アプリのデプロイは Visual Studio Team Services が行うので、Azure に認証済みの接続が必要になります。

15. 続いて、[App Service name] のリストボックス右端の [v] をクリックして、「vsts-handson-0001」を選択します。

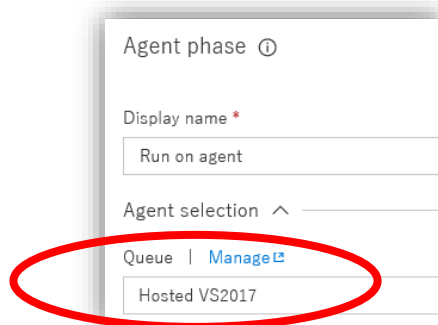


※Visual Studio Team Services のドメインを **vsts-handson-0001** と設定した場合の例です。
各自、Azure App Service Web App を作成したアプリ名に置き換えてください。

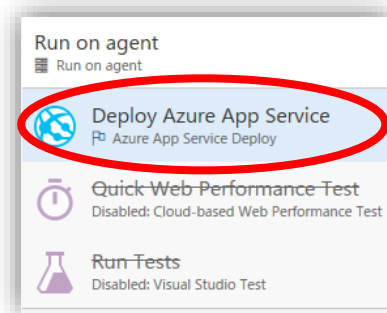
16. 次に、[Run on agent] をクリックします。



17. Agent の設定画面になるので、[Queue] として [Hosted VS2017] が選択されていることを確認します。



18. 次に、タスク一覧から [Deploy Azure App Service] をクリックします。

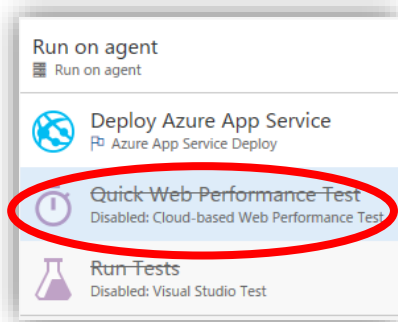


19. Azure へのデプロイ設定画面が表示されるので、パラメーターを以下のように入力します。

パラメーター	説明	今回の設定
Deploy to slot	デプロイするスロットを指定する場合にチェックします。	チェックする
Resource group	Web App があるリソースグループを指定します。	vsts-handson-0001 ※
Slot	対象のスロットを指定します。	staging

※Visual Studio Team Services のドメインを **vsts-handson-0001** と設定した場合の例です。
各自、Azure App Service Web App を作成したリソースグループ名に置き換えてください。

20. 次に、[Quick Web Performance Test] をクリックします。

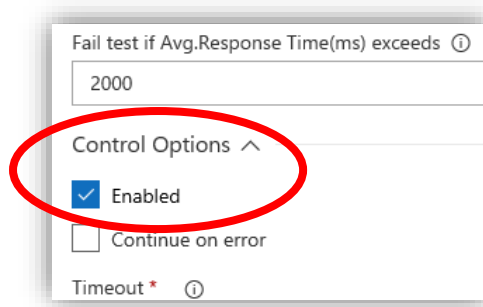


21. ロードテストの設定画面が表示されるので、パラメーターを以下のように入力します。

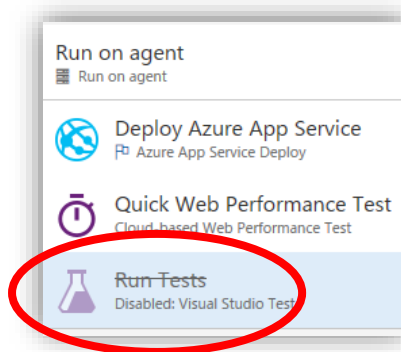
Website Url *	\$(APP_SERVICE_APPLICATION_URL)
Test Name *	LoadTest
User Load *	25
Run Duration (sec) *	60
Load Location ⓘ	Default
Run load test using *	<input checked="" type="radio"/> Automatically provisioned agents <input type="radio"/> Self-provisioned agents
Fail test if Avg.Response Time(ms) exceeds ⓘ	2000

パラメーター	説明	今回の設定
Website Url	テスト対象の URL です。	<規定値>
Test Name	テストの名前を入力します。	LoadTest
User Load	Web アプリにアクセスする仮想ユーザーの数を指定します。	25
Run Duration	ロードテストをかけ続ける時間を指定します。	60
Load Location	仮想ユーザーが、どこから Web アプリにアクセスするかを指定します。	Default
Run load test using	テスト対象がインターネットからアクセス可能な場合は auto を選択します。	Automatically provisioned agents を選択
Fail test if Avg.Response Time(ms) exceeds	応答時間の平均がこの時間を超えるとテスト失敗になります。	2000

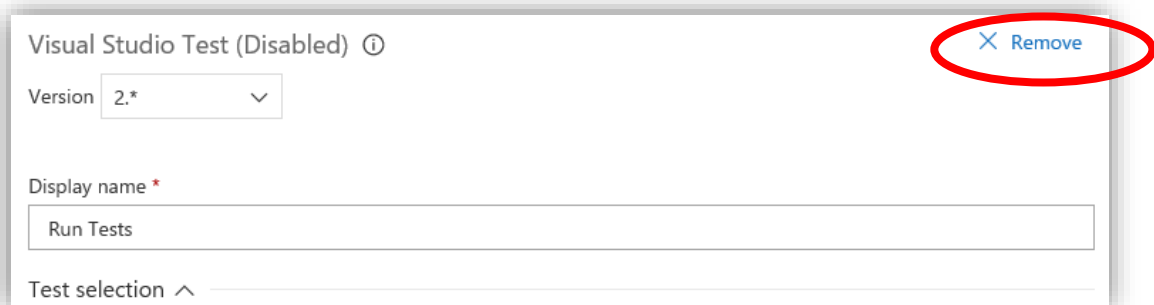
22. Quick Web Performance Test は、デフォルトでは無効（実施しない）になっているので、[Control Options] をクリックし、[Enabled] をチェックして有効にします。



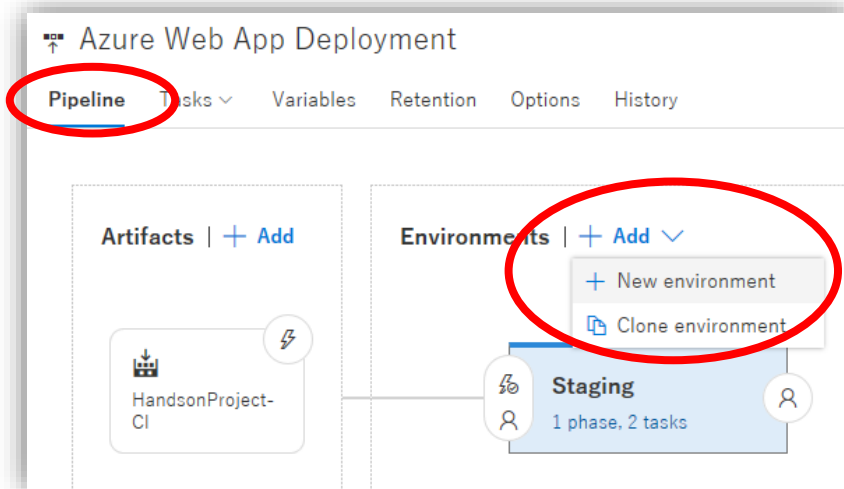
23. 次に、[Run Tests] をクリックします。



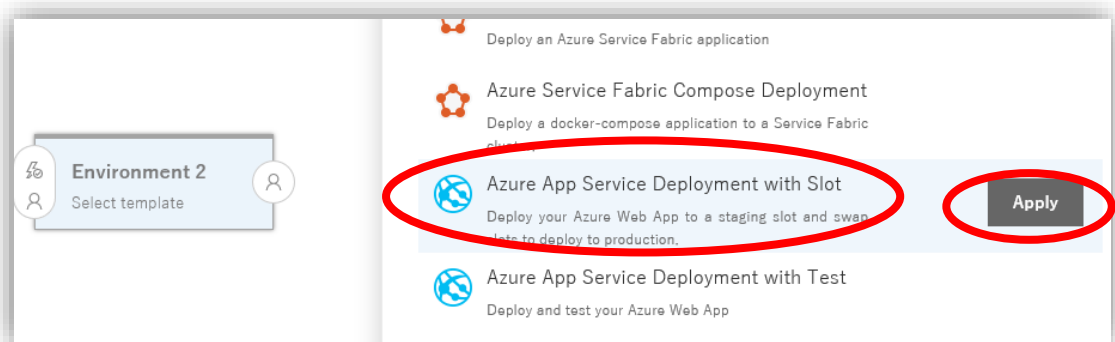
24. 今回このタスクは不要なので、[× Remove] をクリックして削除します。



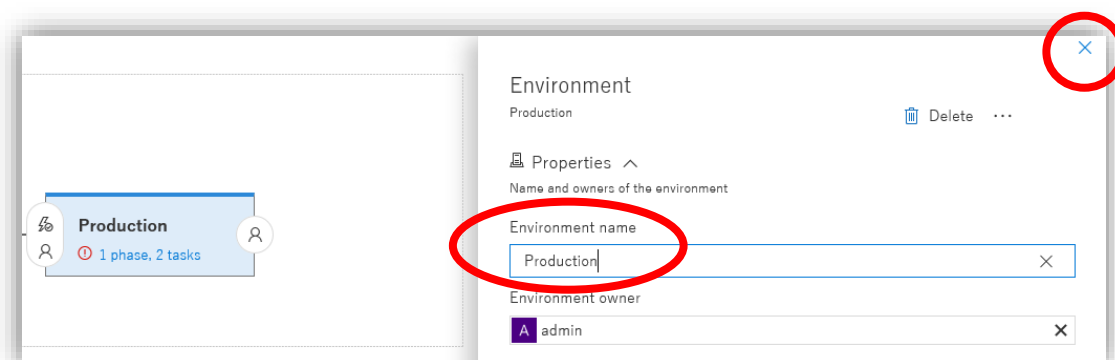
25. 次に、Production 用の環境を作成します。[Pipeline] タブをクリックし、[Environment] の [+Add] – [New environment] をクリックします。



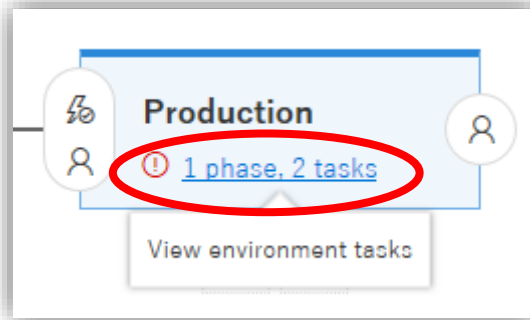
26. 画面右側の [Select a Template] から、[Azure App Service Deployment with Slot] を選択し、[Apply] をクリックします。



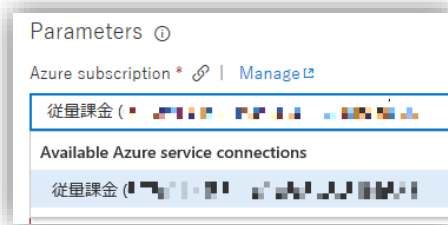
27. 環境の設定画面が表示されるので、[Environment name] に「Production」と入力し、[×] をクリックします。



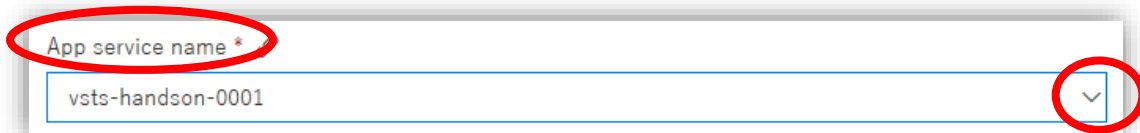
28. 次に、Production 環境で実行するタスクの設定をします。[Production] の [1 phase, 2 tasks] をクリックします。



29. [Azure subscription] に、手順 11 ～ 14 で作成したエンドポイント (Available Azure service connection) を選択します。



30. 続いて、[App Service name] のリストボックス右端の [v] をクリックして、「vsts-handson-0001」を選択します。



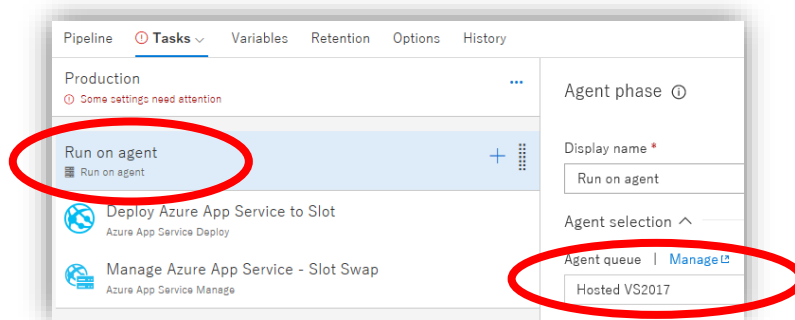
※Visual Studio Team Services のドメインを **vsts-handson-0001** と設定した場合の例です。
各自、Azure App Service Web App を作成したアプリ名に置き換えてください。

31. [App Service name] 設定後、追加のパラメーターが表示されるので、以下のように入力します。

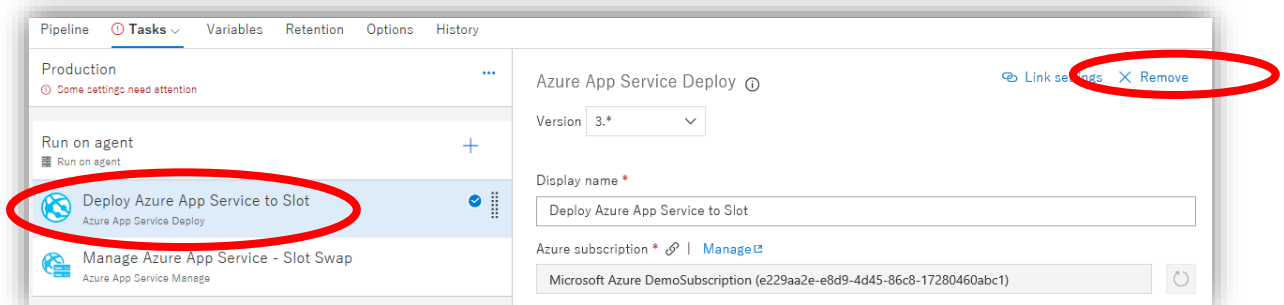
パラメーター	説明	今回の設定
Deploy to slot	デプロイするスロットを指定する場合にチェックします。	チェックする
Resource group	Web App があるリソースグループを指定します。	vsts-handson-0001 ※
Slot	対象のスロットを指定します。	staging

※Visual Studio Team Services のドメインを **vsts-handson-0001** と設定した場合の例です。
各自、Azure App Service Web App を作成したリソースグループ名に置き換えてください。

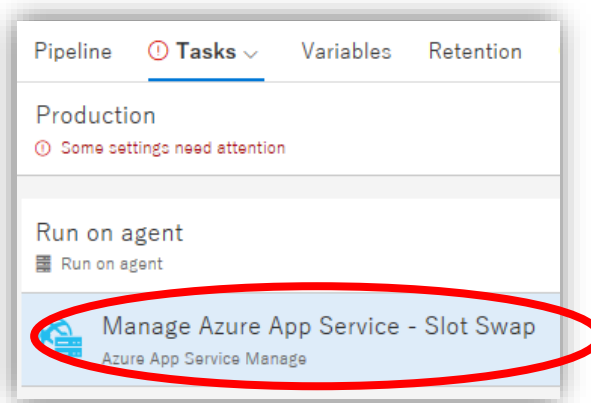
32. 次に、タスク一覧から [Run on agent] をクリックし、[Queue] として [Hosted VS2017] が選択されていることを確認します。



33. 次に、[Deploy Azure App Service to Slot] をクリックします。このタスクは不要なので、画面右の [×Remove] をクリックして削除します。



34. 次に、[Manage Azure App Service – Slot Swap] をクリックします。

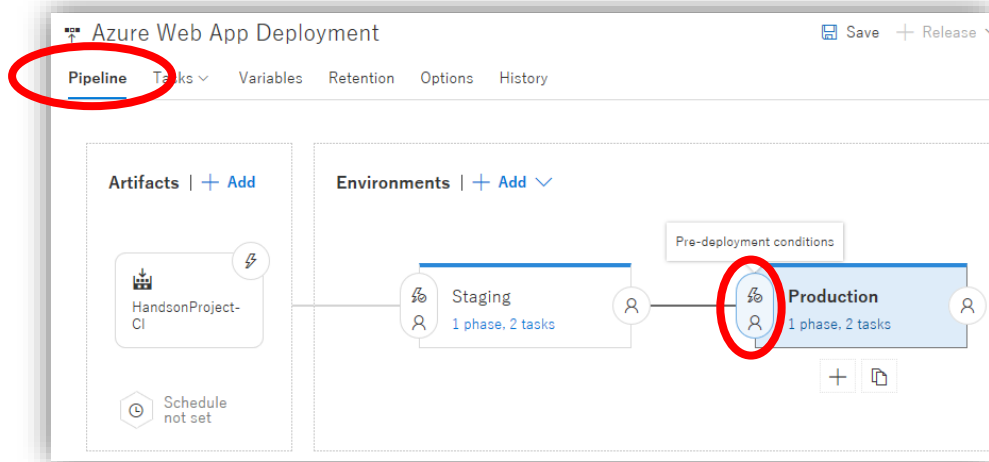


35. [Azure App Service Manage] の各パラメーターはデフォルト値から変更不要です。以下のように入力されていることを確認します。

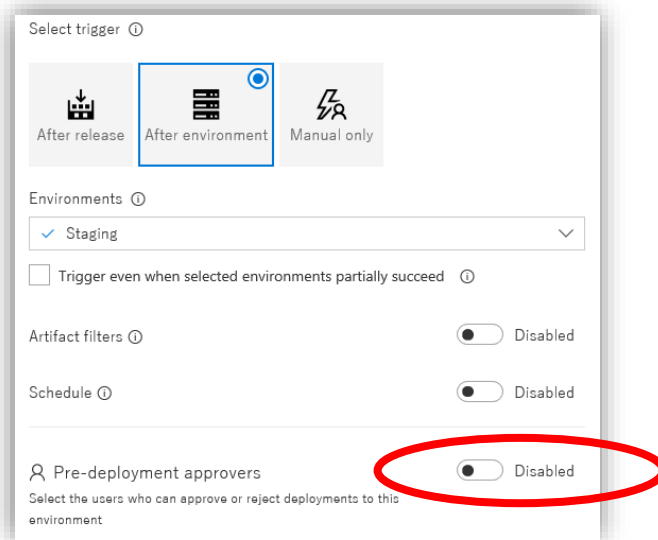
パラメーター	説明	今回の設定
Action	このプロセスが実行するアクションを選択します。	Swap Slots
App Service name	Web App の名前を選択します。	vsts-handson-0001 ※
Resource group	Web App があるリソースグループを指定します。	vsts-handson-0001 ※
Source Slot	スワップ元となるスロットを選択します。	staging
Swap with Production	既定のスロットにスワップするかを設定します。	チェックする

※Visual Studio Team Services のドメインを **vsts-handson-0001** と設定した場合の例です。

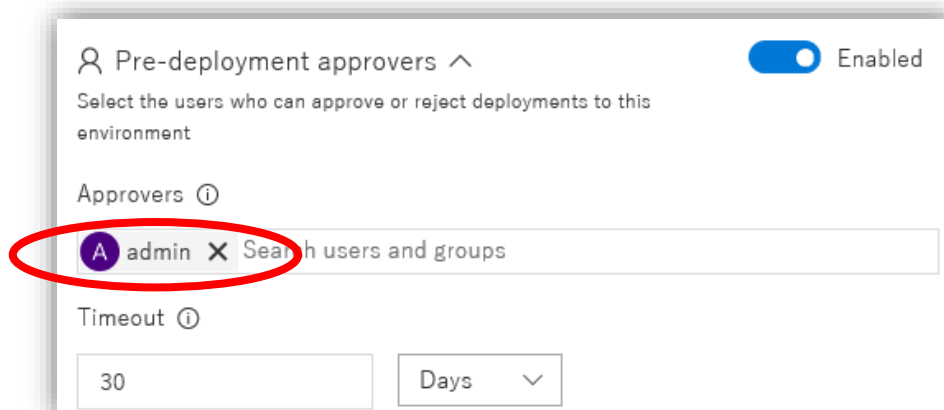
36. 次に、管理者などの特定ユーザーの承認を得ないと、**Production** 環境のデプロイが開始されない設定をします。**[Pipeline]** をクリックし、**[Production]** の **[Pre-deployment conditions]** をクリックします。



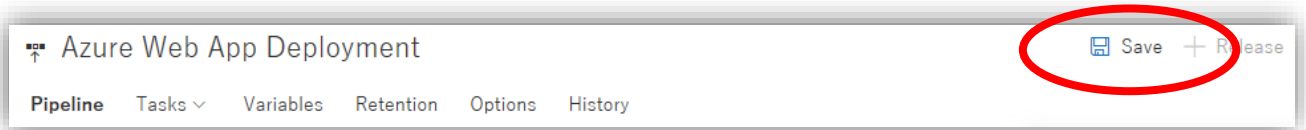
37. **[Pre-deployment approvers]** のスイッチをクリックして、有効にします。



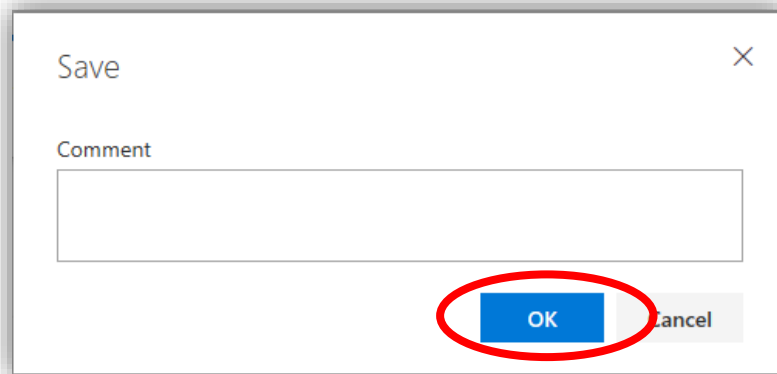
38. デプロイ承認者の設定が表示されるので、**[Approvers]** に現在 Visual Studio Team Services にサインイン中のアカウントを指定します。



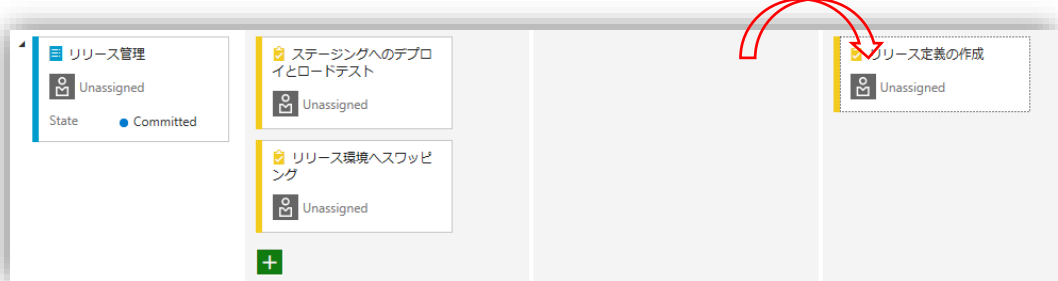
39. これですべての設定が完了したので、[Save] をクリックします。



40. ダイアログが表示されるので、[OK] をクリックします。



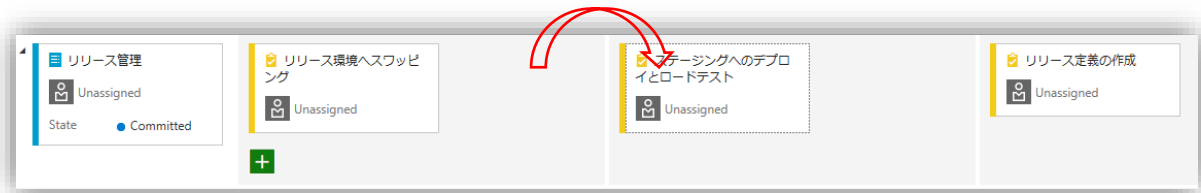
41. 最後に、[Work] - [Backlogs] - [Sprint 1] を開きます。これでリリース定義の作成は完了のため、Task [リリース定義の作成] を [In Progress] から [Done] にドラッグ & ドロップします。



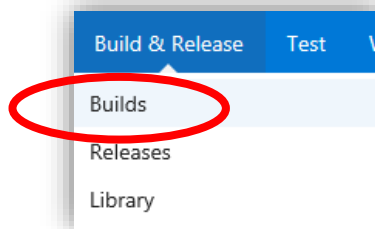
22. ステージングへのデプロイとロードテスト

次の手順では、先ほど作成したリリース定義を使用して、最新のビルドを Azure App Service Web App にデプロイします。まずは、ステージングスロットにデプロイし、その環境でロードテストを行います。

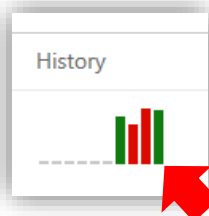
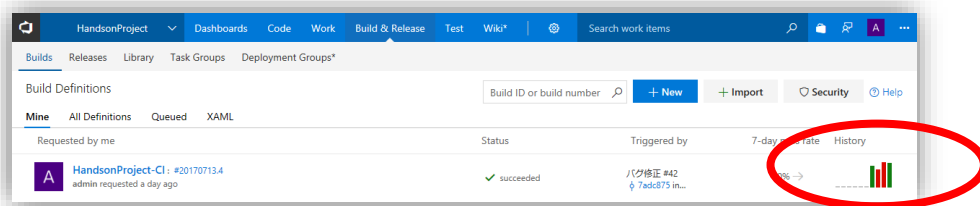
1. [Work] - [Backlogs] - [Sprint 1] を開きます。これから、ステージングへのデプロイとロードテストを行うため、Task [ステージングへのデプロイとロードテスト] を [To do] から [In Progress] にドラッグ & ドロップします。



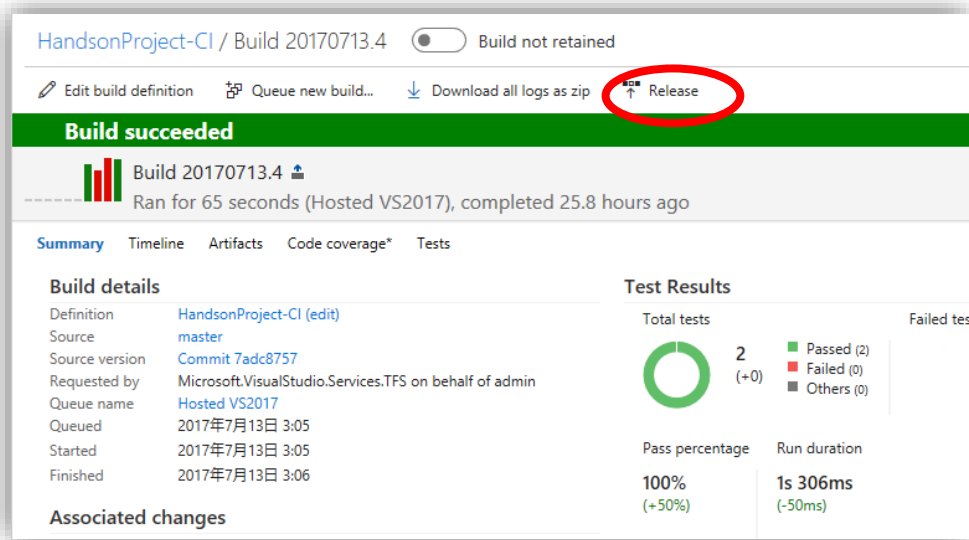
2. メニューバーから、[Build & Release] – [Builds] をクリックします。



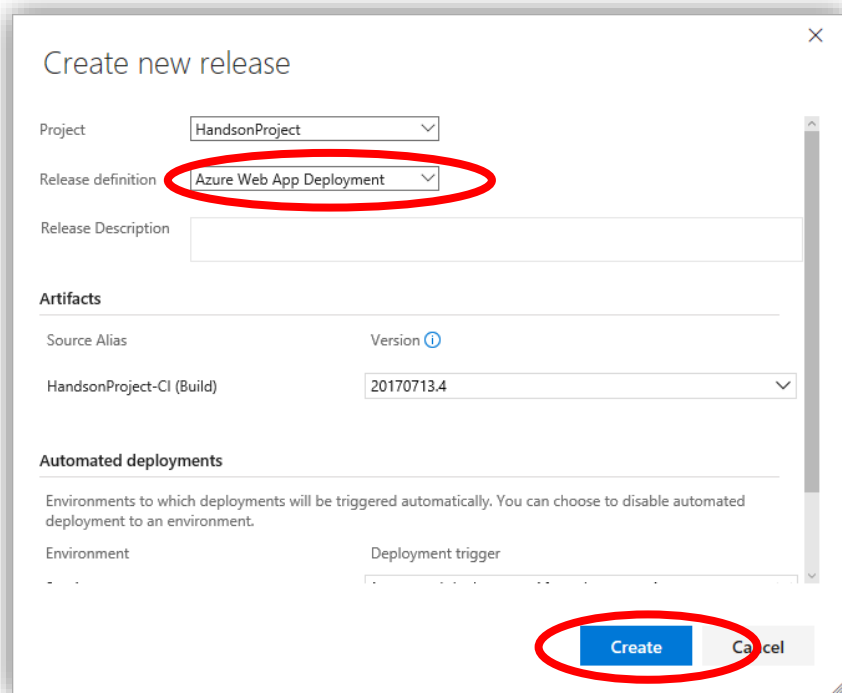
3. ビルドパイプライン [HandsonProject-CI] の [History] から、いちばん右のグラフをクリックします。



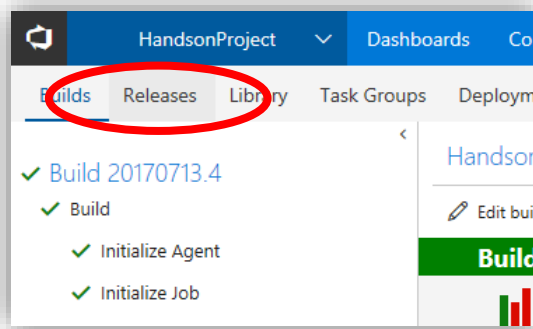
4. 最新のビルドが表示されるので、[Release] をクリックします。



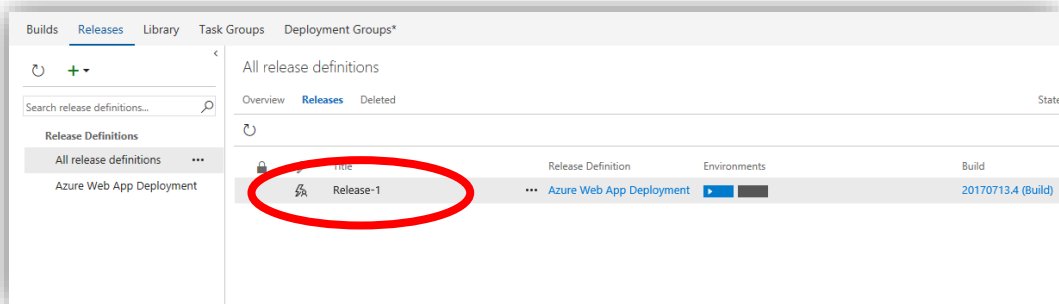
5. 新しいリリース作成のダイアログが表示されるので、[Release definition] が先ほど作成した [Azure Web App Deployment] になっていることを確認し、[Create] をクリックします。



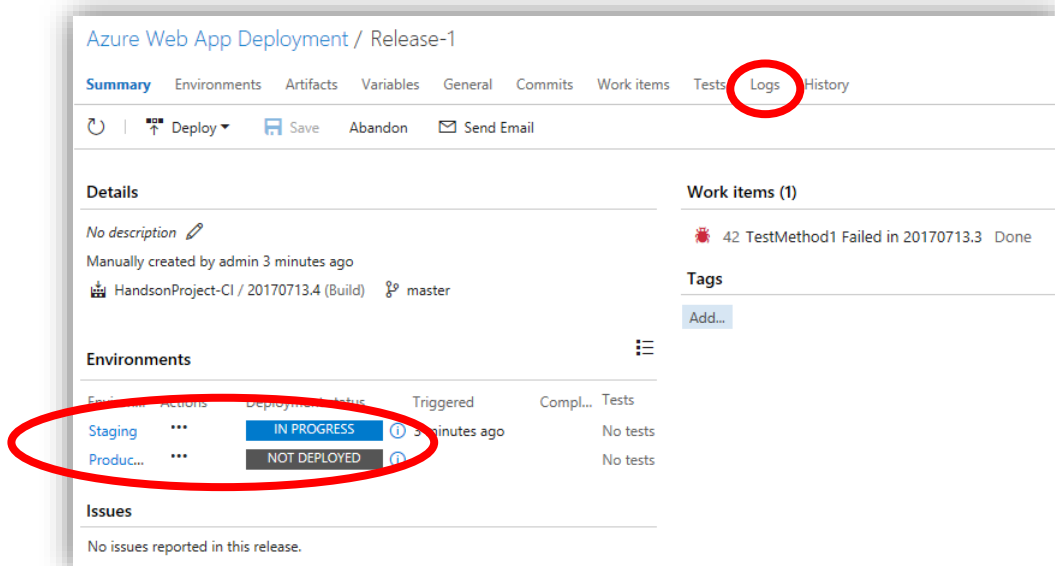
6. 作成と同時にリリースが開始されるので、状態を確認します。[Releases] タブをクリックします。



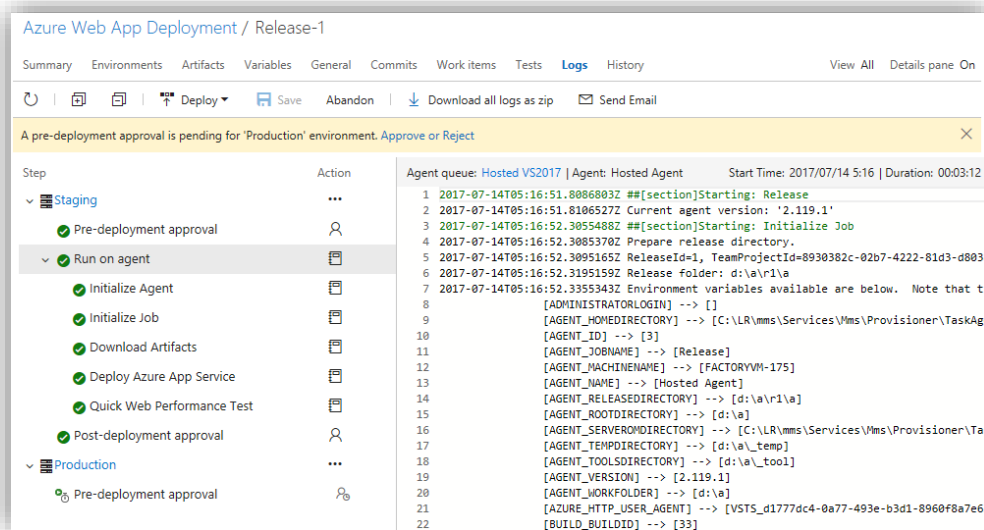
7. [Release-1] が作成されていることを確認し、[Release-1] をクリックします。



8. Summary が表示されるので、[Staging] の環境が [In PROGRESS] (デプロイ中) であることを確認します。続いて、[Logs] タブをクリックします。



9. [Logs] 画面では、各プロセスをクリックすると、それぞれのログを表示することができます。ここでは、主なプロセスの役割を確認します。



● Initialize Agent

ホストされた Agent が、リリースを実行する準備をします。

● Initialize Job

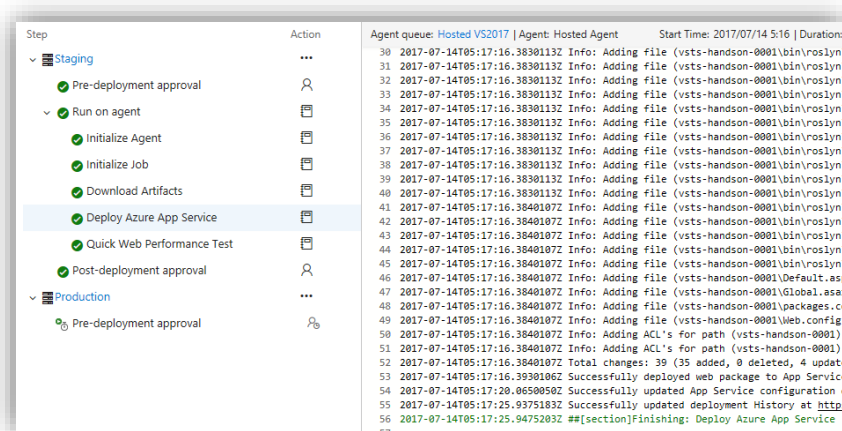
Agent が Job を実行する準備をします。

● Download Artifacts

デプロイする Artifact を取得します。

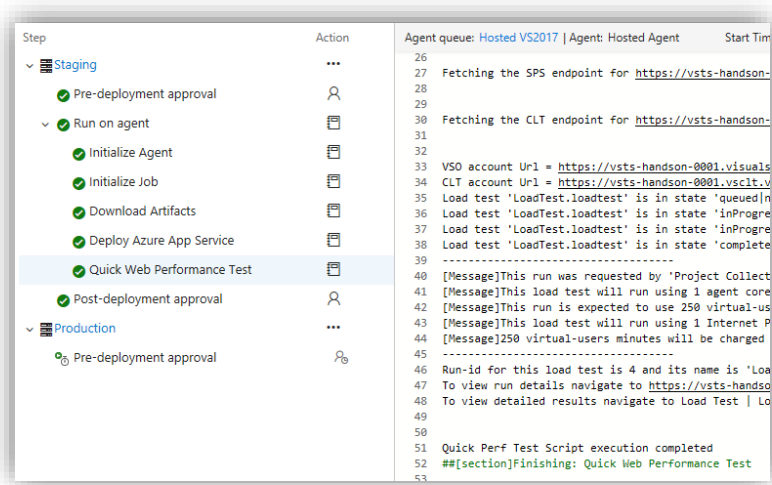
● Deploy Azure App Service

Azure App Service へアプリをデプロイします。

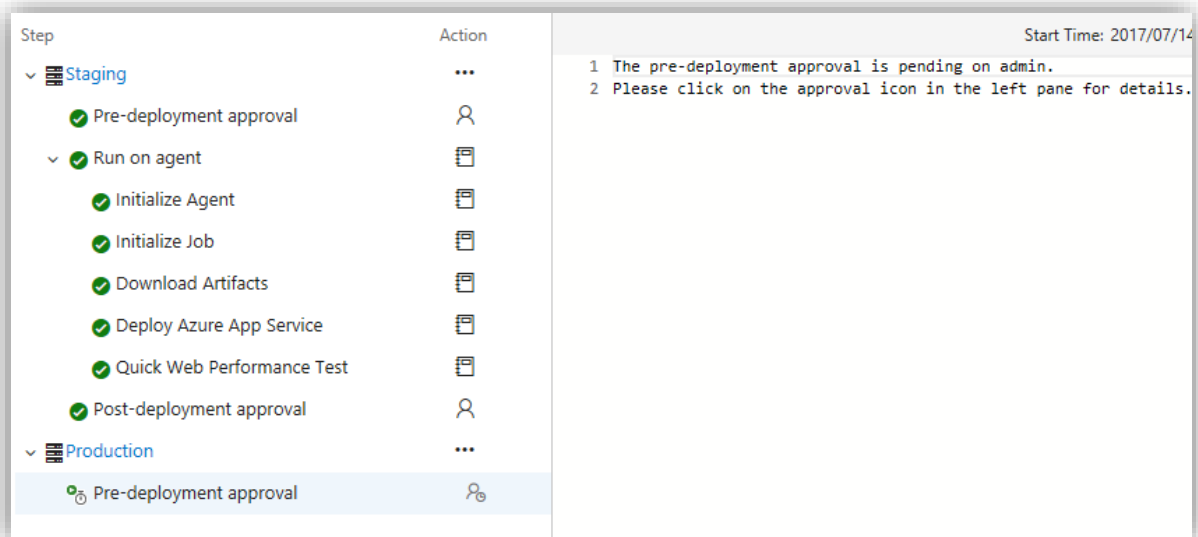


●Quick Web Performance Test

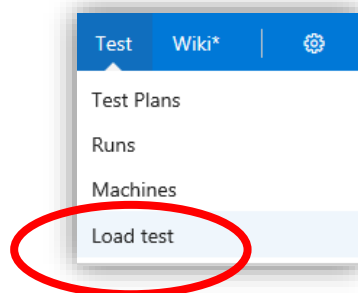
ロードテストを実行します。



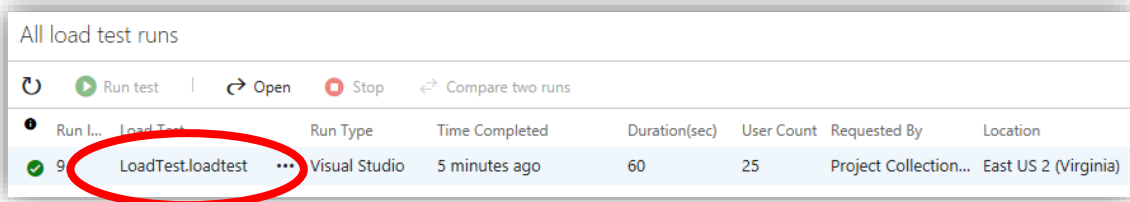
10. [Staging] 環境のデプロイは問題なく完了しました。[Production] 環境へのデプロイは、管理者の認証が必要なので、現在は [Staging] 完了でストップしています。



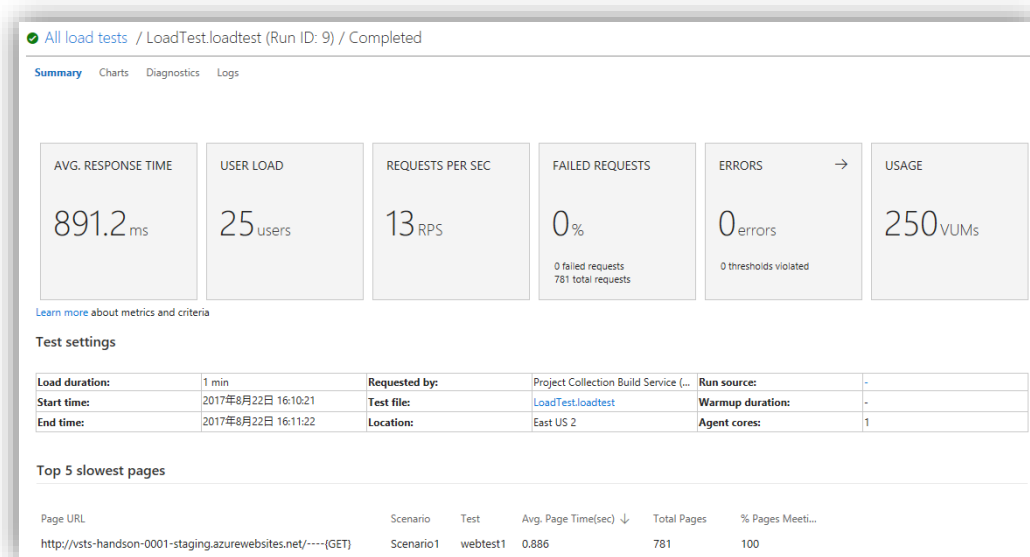
11. 続いて、ロードテストの結果を確認します。メニューバーから、[Test] – [Load test] をクリックします。



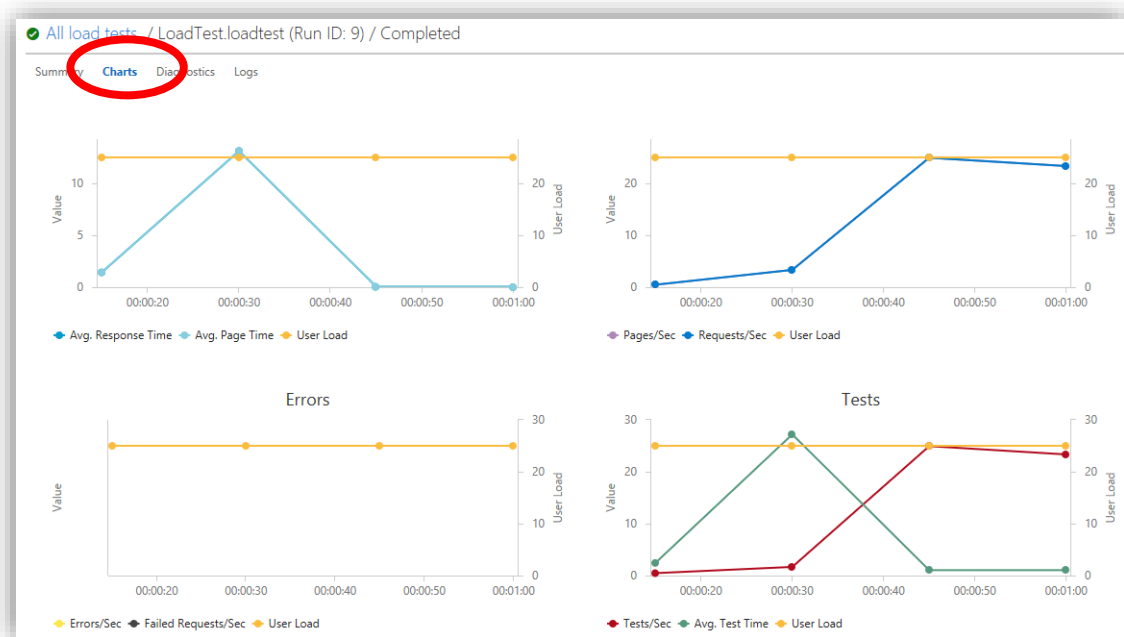
12. ロードテストの結果一覧が表示されるので、[LoadTest.loadtest] をダブルクリックします。



13. ロードテストの結果が表示されます。以下の結果では、[AVG.RESPONSE TIME] が指定の 2000ms より小さいため、パス (合格) したと判定されています。



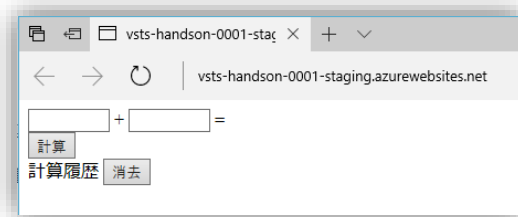
14. [Charts] タブをクリックすると、パフォーマンスやエラーなどのグラフを参照できます。



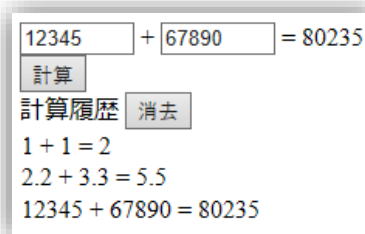
15. 続いて、ステージングにデプロイされたアプリを確認します。

Web ブラウザーで <http://vsts-handson-0001-staging.azurewebsites.net/> を開きます。

※この URL は、アプリ名が **vsts-handson-0001** の場合の URL です。アプリ名に指定した値に置き換えてください。

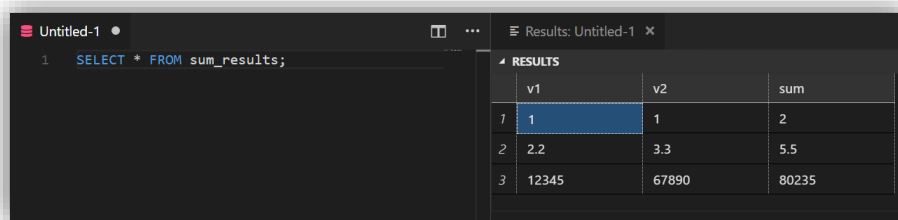
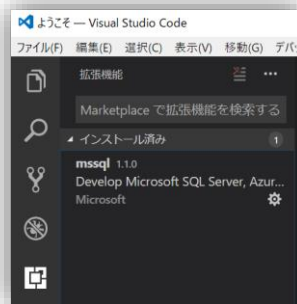


16. いくつか計算して、複数の履歴を残しておきます。この履歴は、ステージング用のデータベース **handson-db-staging** に記録されます。



ワンポイント

SQL Database を GUI で管理するツールは色々ありますが、Visual Studio Cod に “mssql” の拡張機能をインストールすれば、Visual Studio Code でもテーブルの参照やクエリの実行ができるようになります。

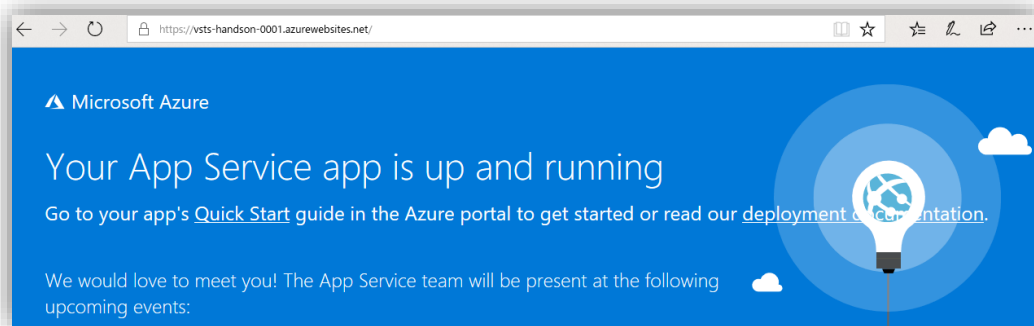


詳しくは、以下を参照してください。

<https://docs.microsoft.com/ja-jp/sql/linux/sql-server-linux-develop-use-vscode>

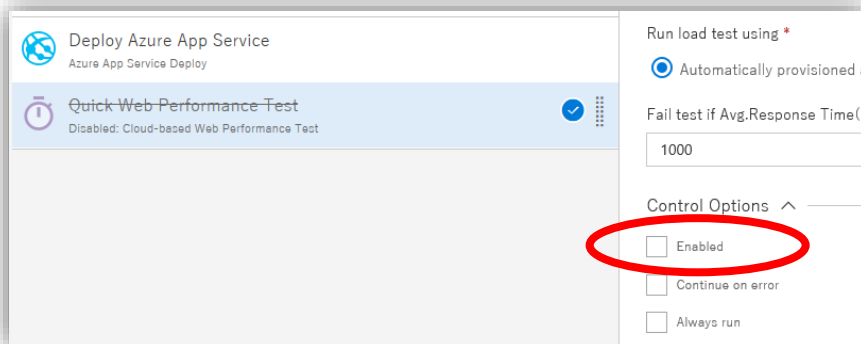
17. リリース環境の URL <http://vsts-handson-0001.azurewebsites.net/> も開いてみます。こちらにはデプロイされていないので、デフォルトのページのままになっています。

※この URL は、アプリ名が **vsts-handson-0001** の場合の URL です。アプリ名に指定した値に置き換えてください。

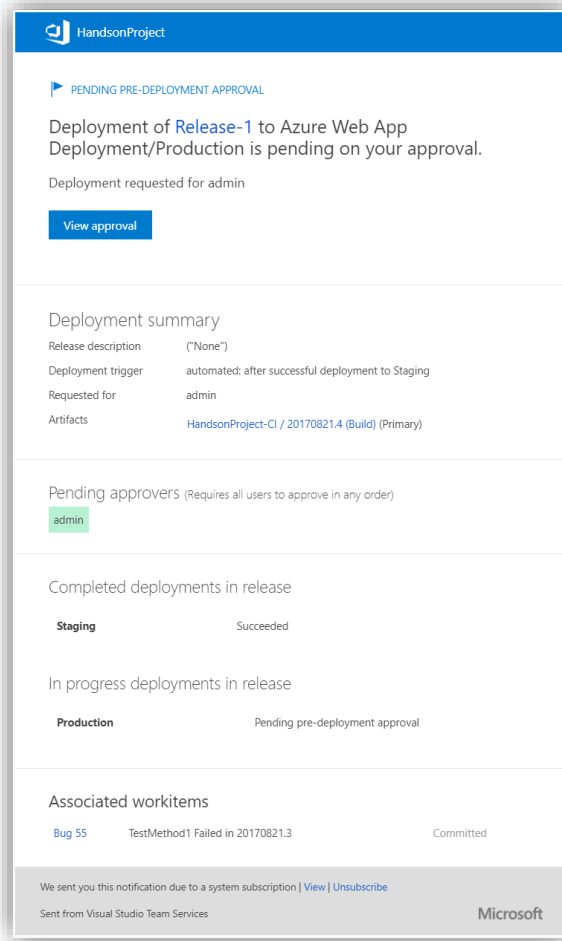


ワンポイント

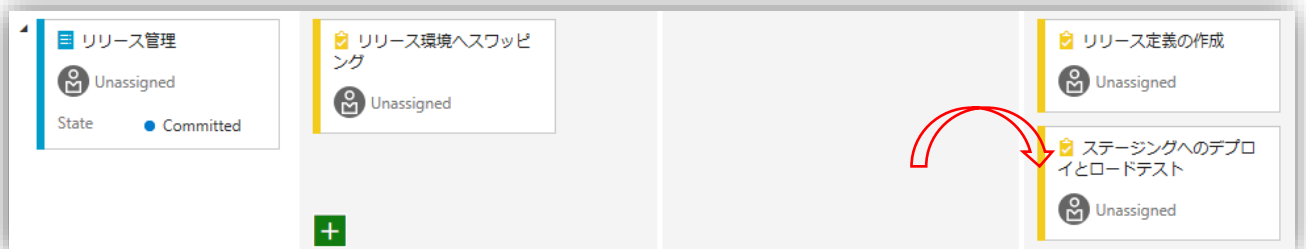
現在のリリース定義では、リリースを行うたびにロードテストが実行されます。毎回ロードテストを行う必要はないので、テストを行いたい時だけ [Quick Web Performance Test] の [Enabled] にチェックをします。



18. 続いて、[Production] 環境へのデプロイが承認待ちであることを通知するメールが届いていることを確認します。

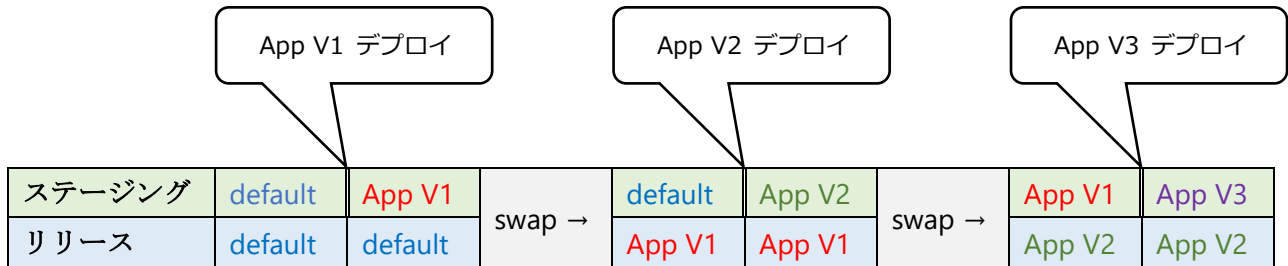


19. 最後に、[Work] - [Backlogs] - [Sprint 1] を開きます。これでステージングへのデプロイとロードテストは完了のため、Task [ステージングへのデプロイとロードテスト] を [In Progress] から [Done] にドラッグ & ドロップします。



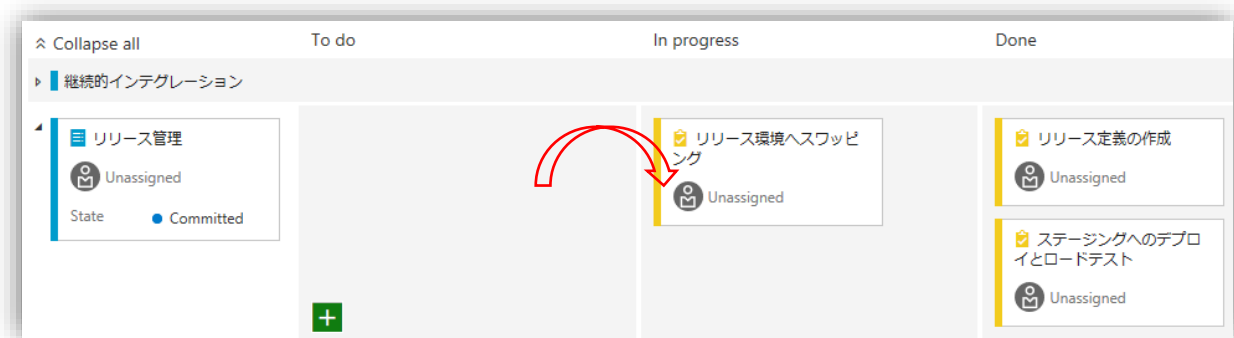
23. リリース環境へスワッピング

ビルドしたアプリは、現在ステージングにて動作しています。本来ならばここで手動（または自動）による各種テストを行い、管理者の承認を得て、リリース（運用）環境に切り替えることとなりますが、本手順書ではテストはパスしたものとし、管理者の承認作業とスワッピング（交換）の確認のみ行います。

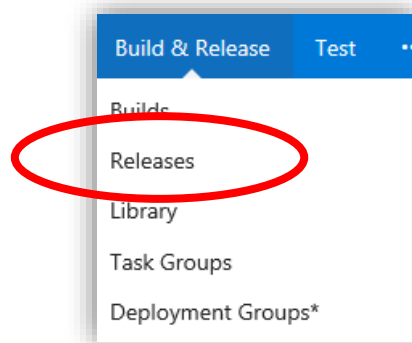


次の手順では、ステージングにデプロイしたアプリを、リリース環境にスワッピングします。

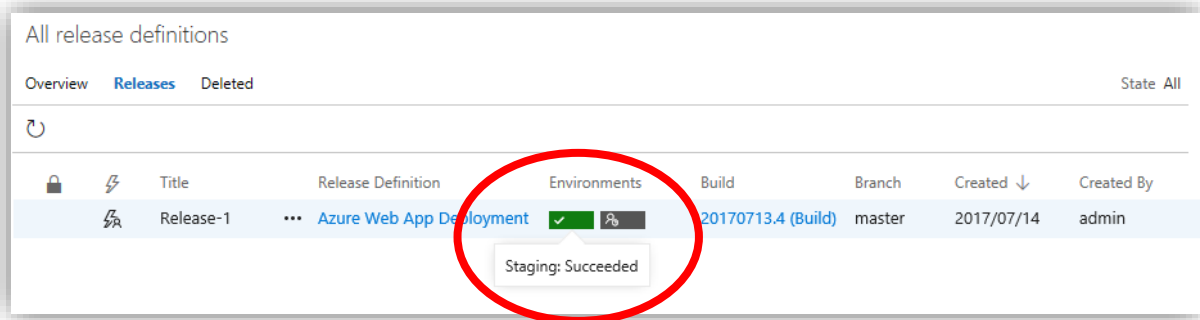
1. [Work] - [Backlogs] - [Sprint 1] を開きます。これから、リリース環境へスワッピングを行うため、Task [リリース環境へスワッピング] を [To do] から [In Progress] にドラッグ & ドロップします。



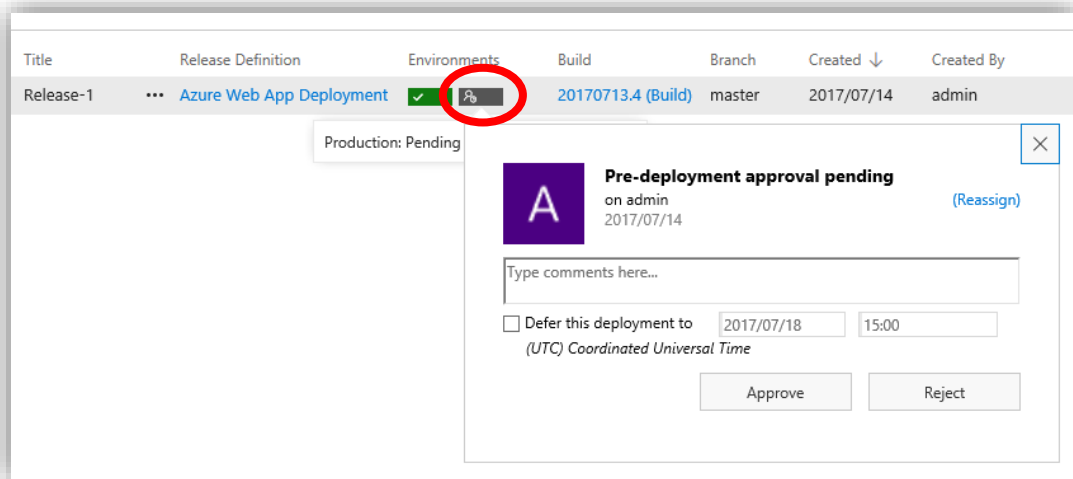
2. [Build & Release] – [Releases] をクリックします。



3. [All release definitions] に、先ほどステージング環境 (Staging) にデプロイを行った [Release-1] が表示されます。ステージング環境へのデプロイは成功しており、リリース環境 (Production) へのデプロイは管理者の承認待ち状態になっています。

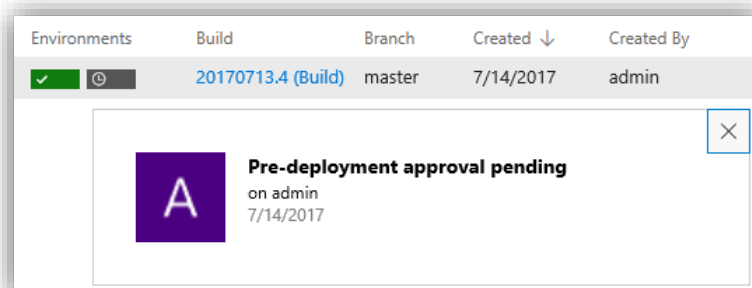


4. [Environments] から [Production] をクリックすると、承認のメッセージボックスが表示されます。

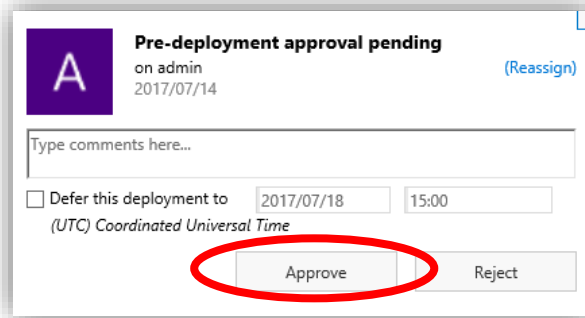


ワンポイント

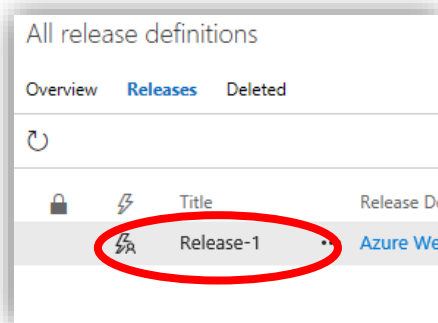
承認権限がないユーザーでサインインしている場合は、以下のような表示になります。



5. [Production] の実行を許可します。[Approve] をクリックします。



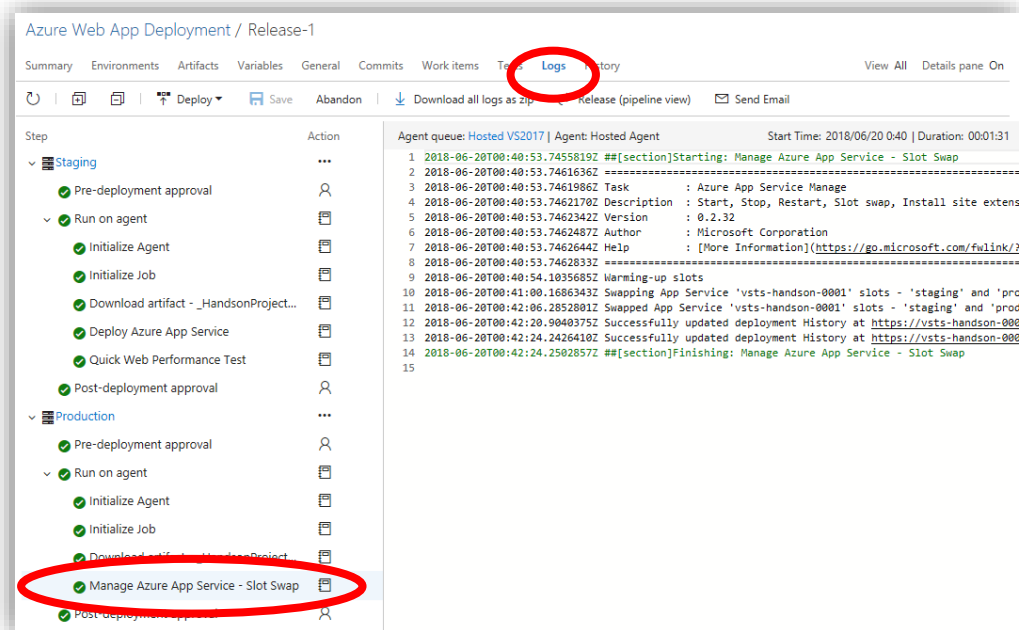
6. これで [Production] へのスワッピングが開始されます。状態を確認するため、[Release-1] をクリックします。



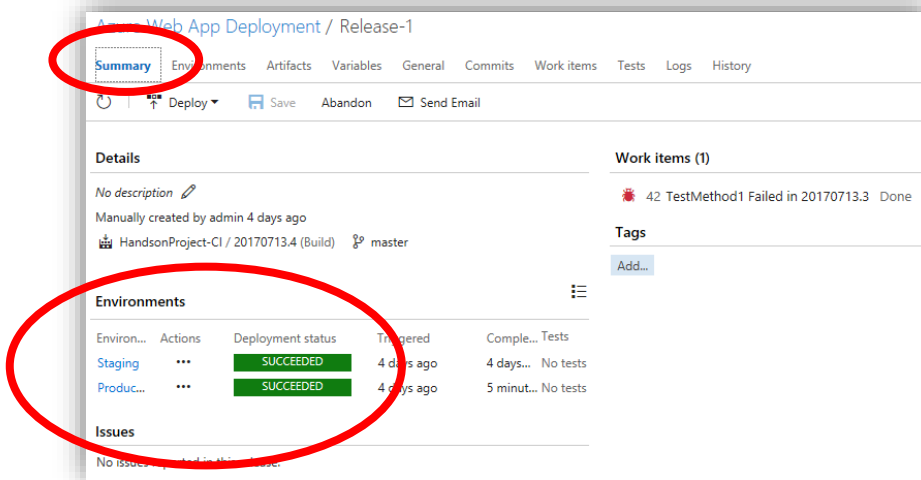
7. [Logs] 画面で、[Manage Azure App Service – Slot Swap] プロセスの役割を確認します。

● Swap Slots

スワッピングを実行します。

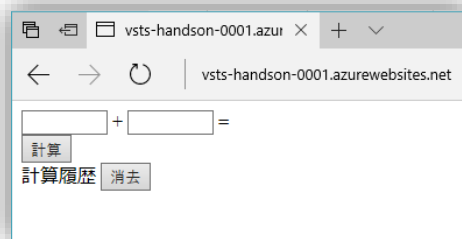


8. [Summary] をクリックします。[Staging] と [Production] の作業が完了したことがわかります。



9. 続いて、リリース環境にスワッピングされたアプリを確認します。

Web ブラウザーで <http://vsts-handson-0001.azurewebsites.net/> を開きます。この URL は、スワッピング前はデフォルトのページでしたが、今はアプリに入れ替わっています。※この URL は、アプリ名が **vsts-handson-0001** の場合の URL です。アプリ名に指定した値に置き換えてください。

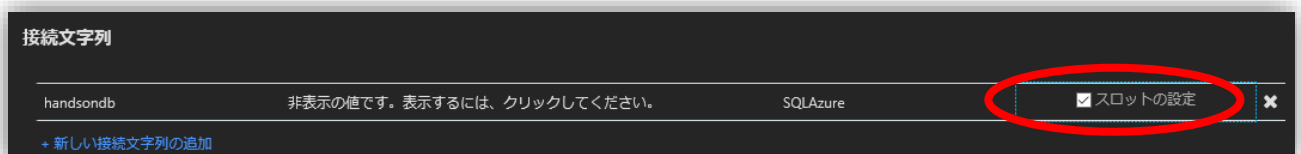


ここで、計算履歴が表示されないことを確認します。

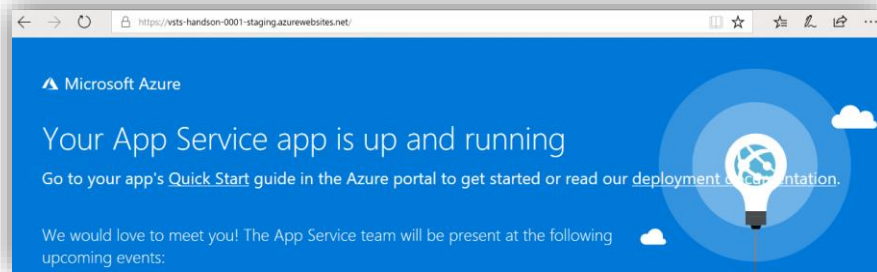
スワッピングすることにより、これまでステージング環境だったものがリリース環境になるよう交換されるわけですが、データベースの接続文字列は各スロット固定なので、交換されません。つまり、何度スワッピングしても、リリース環境は必ず **handson-db** を、ステージング環境は必ず **handson-db-staging** を参照することになります。

ワンポイント

スロット固定する設定は Web App の [接続文字列] で行っています。



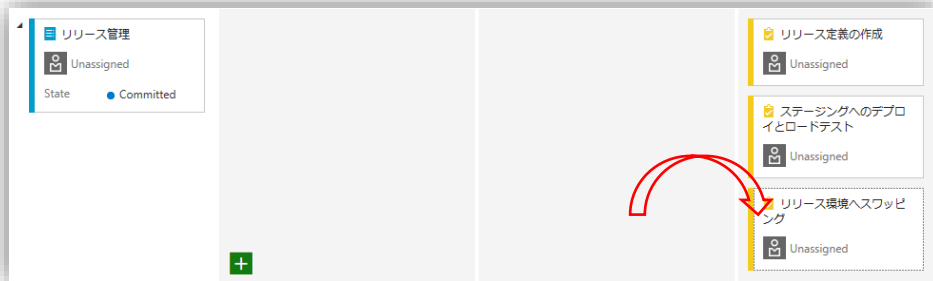
10. ステージング環境の URL <http://vsts-handson-0001-staging.azurewebsites.net/> も開いてみます。この URL は、スワッピング前はアプリでしたが、今はデフォルトのページに入れ替わっています。※この URL は、アプリ名が **vsts-handson-0001** の場合の URL です。アプリ名に指定した値に置き換えてください。



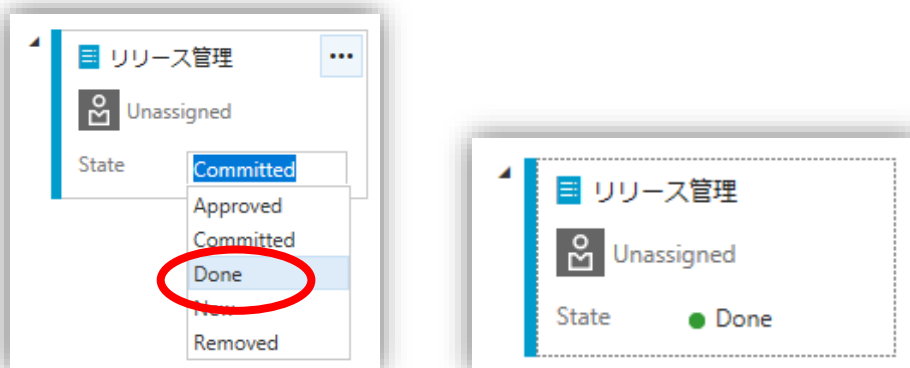
ワンポイント

以後は、コーディング → リモトリポジトリへ **push** → **[自動]ビルド & テスト** → **[手動]ステージング環境へデプロイ (& ロードテスト)** → **[手動承認]リリース環境へスワッピング** の繰り返しになります。

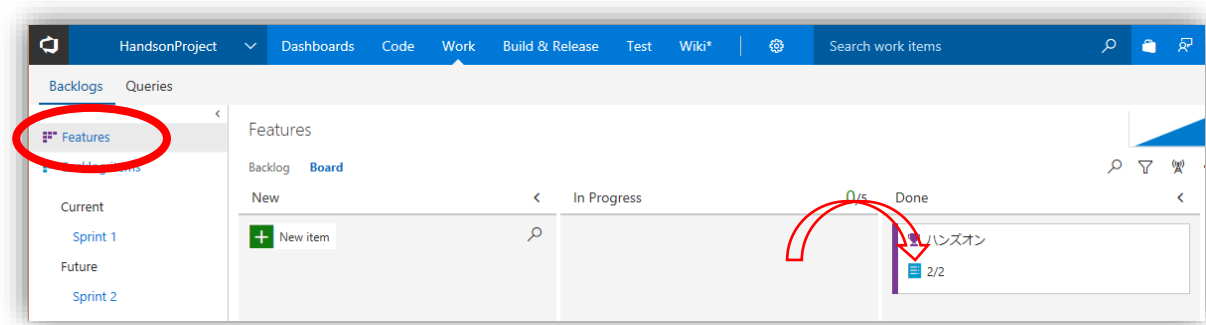
11. 最後に、[Work] - [Backlogs] - [Sprint 1] を開きます。これでリリース環境へスワッピングは完了のため、Task [リリース環境へスワッピング] を [In Progress] から [Done] にドラッグ & ドロップします。



12. これで、[リリース管理] の Task はすべて完了したので、ステータスを [Committed] から [Done] に変更します。



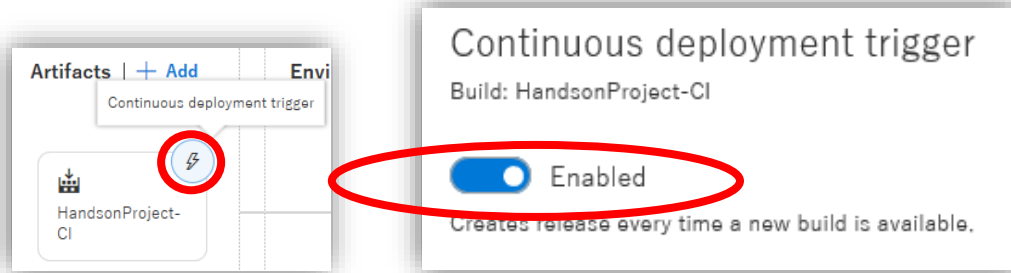
13. 画面左の [Features] をクリックします。これで [ハンズオン] のすべての作業が完了したので、[In Progress] から [Done] にドラッグ & ドロップします。



ワンポイント

本自習書では、ビルド成功後のステージングへのデプロイは手動で行っていますが、ここの繋ぎも自動化することができますので、簡単に説明します。

1. リリース定義の [Artifacts] - [Build] の右上にある稲妻アイコンをクリックし、[Continuous deployment trigger] のスイッチを Enable にします。その後、[Save] をクリックして保存します。



2. Visual Studio Code で Default.aspx.cs を開き、12 行目の Button1_Click 関数を以下のように変更 (青色の部分を追加) して保存します。その後、「エラー処理追加」という Commit Message で commit → push します。

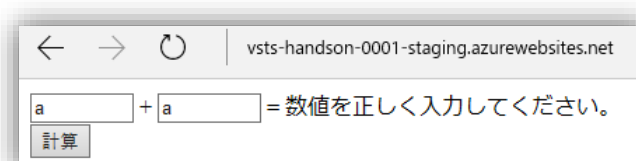
```
protected void Button1_Click(object sender, EventArgs e) {
    try {
        float val = this.Calculate(this.TextBox1.Text, this.TextBox2.Text);
        this.Label1.Text = string.Format("{0}", val);
    } catch (FormatException) {
        this.Label1.Text = "数値を正しく入力してください。";
    }

    // 履歴を残します。
}
```

3. ステージングへのデプロイまで自動で行われます。

	🔒 ⚡	Title	Release Definition	Environments
↑		Release-2	... Azure Web App Deployment	✓ 🔄
🔄		Release-1	Azure Web App Deployment	✓ ✓

4. ステージングの URL を開きます。アプリのテキストボックスに数値以外の値を入力し計算すると、エラーメッセージが表示されることを確認します。



参考文献

この自習書の作成に当たり、参考にさせていただいた Web サイトをまとめます。

24. Visual Studio Team Services に関する情報の入手元

Visual Studio Team Services に関する最新の情報は、次の Web サイトから入手できます。

- Visual Studio Team Services 公式サイト

製品情報、価格、技術情報など、Visual Studio Team Services に関するすべての情報への入口です。

<https://www.microsoft.com/ja-jp/dev/products/visual-studio-online.aspx>

- Azure の公式サイトから

こちらのサイトも上記と同じく Visual Studio Team Services のコンテンツが充実しているほか、無償利用の開始もここから行います。

<https://azure.microsoft.com/ja-jp/services/visual-studio-team-services/>

- visualstudio.com のドキュメント

すべて英語ですが、Visual Studio Team Services についての技術的な解説やチュートリアルなどが掲載されています。

<https://www.visualstudio.com/ja-jp/docs/devops-alm-overview>

<https://www.visualstudio.com/ja-jp/docs/build/overview> (Build & Release)

<https://www.visualstudio.com/ja-jp/docs/work/overview> (Agile tools)

<https://www.visualstudio.com/ja-jp/docs/git/overview> (Git)

<https://www.visualstudio.com/ja-jp/docs/test/index> (Test)

25. その他の参考文献

●Git のドキュメント

Git に関するドキュメントです。Book では、Git の仕組みやワークフローなどが記載されています。Reference では、git コマンドの使い方や、設定について掲載されています。

<https://git-scm.com/book/en/v2> (Book)

<https://git-scm.com/docs> (Reference)

●Visual Studio Code のドキュメント

Visual Studio Code に関するドキュメントです。基本的な使用方法や拡張方法などが掲載されています。

<https://code.visualstudio.com/docs>

<https://code.visualstudio.com/docs/editor/versioncontrol> (Git 連携)

●Web Apps のドキュメント

Azure App Service Web App に関するドキュメントです。技術的な解説やチュートリアルなどが掲載されています。

<https://docs.microsoft.com/ja-jp/azure/app-service-web/>

<https://docs.microsoft.com/ja-jp/azure/app-service-web/web-sites-staged-publishing> (Staging 設定)

<https://blogs.msdn.microsoft.com/benjaminperkins/2017/09/04/database-connection-string-when-swapping-between-app-servers-slots/> (接続文字列固定のスワッピング)

●SQL Database のドキュメント

Azure SQL Database に関するドキュメントです。技術的な解説やチュートリアルなどが掲載されています。

<https://docs.microsoft.com/ja-jp/azure/sql-database/>