Microsoft Dynamics 365 Customer Engagement

DevOps 1 Day with Lab

Lab 3: Azure Release Pipelines

Student Lab Manual

Conditions and Terms of Use

Copyright and Trademarks

Table of Contents

Introduction 3
Create a service connection 3
Update pipeline to include a deployment job 4
Option 1: Update the pipeline using Task Assistant 4
Option 2: Update the pipeline in VS Code 7
Conclusion 8

Introduction

In this lab, we will extend our pipeline to take the previously created build artifacts and deploy it into our production environment.

Create a service connection

To connect to your Dynamics 365 Customer Engagement, using the Power Platform Build Tools, you will need to create a service connection.

1. Navigate to the contoso-university project in your Azure DevOps organization at https://dev.azure.com/.
2. Click Project settings.
3. Click Service connections (underneath **Pipelines**).
4. Click Create service connection.
5. Click Generic.
6. Enter the following information in the form that pops up:

   Server URL: Copy/paste the URL to your production environment in the Power Platform admin portal (https://admin.powerplatform.microsoft.com/).

   1. In the admin portal on the Environments page, click the name of your environment.
   2. In the details pane copy the Environment URL.

   Username (optional): Your fully qualified username (e.g.

jane.doe@contosouniversity.onmicrosoft.com)
Password/Token Key (optional): Your password
Service connection name: production

7. Click Save.

Update pipeline to include a deployment job

With the service connection created, we can now add a deployment job to our pipeline, we have two options to accomplish this task:

- **Option 1:** Update the pipeline using Task Assistant.
  This option is recommended for non-developers.
- **Option 2:** Update the pipeline in VS Code.
  This option is recommended for developers.

Option 1: Update the pipeline using Task Assistant

Though this step will require some manual coding, you can still use the task assistant to alleviate some of the difficulty of typing each job by hand.

1. Click Pipelines.
2. Click on your pipeline (contoso-university).
3. Click Edit.
4. First, we will need to move all of our current steps into a single job. On line 3 (the line immediately before `pool:` manually type the following two lines:

   ```
   jobs:
   - job: Pack
   ```

5. Place your cursor at the very beginning of line 5 (now `pool:`), hold Ctrl+Alt+Shift+Down until your cursor stretches to the bottom of the file. Press space twice to insert two spaces on the beginning of each line.

   This indentation places all of the current tasks we've defined into the `Pack` job.

6. Click on line 21 and press Enter to add a new line. Delete the whitespace and manually type or copy and paste the following block of code:

```
- deployment: Deploy
  dependsOn: Pack
  condition: succeeded('Pack')
  pool:
    vmImage: 'windows-latest'
  environment: 'production'
  strategy:
    runOnce:
      deploy:
        steps:
```

   Make sure you have the additional line with eight spaces after `steps:`.

   Here, we're using `deployment` in place of `job` to indicate that this is a deployment job.

   The `dependsOn` property tells the agent that this job (`Deploy`) will not run until `Pack` completes. The `condition` property goes further to insist that this job will not run unless `Pack` completed successfully (that is to say it did not fail or get canceled). This is important as we only want to import artifacts that were built successfully.

   We set the environment to `'production'` to keep track of what build was deployed to each environment.

   We are using the `runOnce` strategy for now because it is the simplest strategy and we are only deploying to a single environment.

   The `deploy` property is one of several lifecycle hooks that compose the RunOnce strategy. For our simple use case, it's all we need.

7. In the task assistant, search for and select "Power Platform Tool Installer" and click Add and manually insert a new line. Note that we need to run this task again because now that we're running from within a new job, we are running on a new container that doesn't have any of the tooling we installed in the Pack job.

8. In the task assistant, search for and select "Power Platform Import Solution."

   a. Enter the following values for the parameters:
      Authentication Type: Username/password (no MFA support)
      Service Connection: Select *Production* from the dropdown
      Solution Input File: $(Pipeline.Workspace)/packed-solution/contoso_university_core_managed.zip

   b. Click Add and manually insert a new line.

Note: In normal jobs, you need to explicitly download any artifacts that you want to use, but deployment jobs will automatically download every artifact that was generated in the pipeline and store each one in a folder in $(Pipeline.Workspace), which is why we can access the deployable `.zip` file in $(Pipeline.Workspace) here.

9. Click Save, leave the default values, and then click Save again.

You do not need to click anything additional to run this pipeline. Because the pipeline is configured to run whenever code is committed to the master branch, and because the pipeline is itself code, once you clicked Save, the pipeline should be triggered to run automatically.

You can confirm this by clicking ← and then noting that the pipeline is now running.

#20191115.1 Update pack-solution.yml for Azure Pipelines
Individual CI master 840566d
Just now
53s

The import and publish will take a few minutes, but once it's done, proceed to the Conclusion of this lab.

Option 2: Update the pipeline in VS Code

Open pack-solution.yml and replace its contents with the following code:

```
trigger:
- master

jobs:
- job: Pack
  pool:
    vmImage: 'windows-latest'
  steps:
  - task: PowerPlatformToolInstaller@0
  - task: PowerPlatformPackSolution@0
    inputs:
      SolutionSourceFolder: 'contents'
      SolutionOutputFile: 'packed-solution/contoso_university_core.zip'
      SolutionType: 'Both'
  - task: PublishPipelineArtifact@1
    inputs:
      targetPath: 'packed-solution'
      artifact: 'packed-solution'
      publishLocation: 'pipeline'
- deployment: Deploy
  dependsOn: 'Pack'
  condition: succeeded('Pack')
  pool:
    vmImage: 'windows-latest'
  environment: 'production'
  strategy:
    runOnce:
      deploy:
        steps:
        - task: PowerPlatformToolInstaller@0
        - task: PowerPlatformImportSolution@0
          inputs:
            AuthenticationType: 'PowerPlatformEnvironment'
            PowerPlatformEnvironment: 'production'
            SolutionInputFile: '$(Pipeline.Workspace)/packed-
solution/contoso_university_core_managed.zip'
            AsyncOperation: true
            MaxAsyncWaitTime: '240'
```

For detailed explanations of each block, see Option 1.

Now, commit and push your changes:

```
git add .
git commit -m "added deployment job to pipeline.yml"
git push
```

Once your push hits the remote repository, the pipeline will trigger on its own due to the trigger configuration we've defined in this YAML file.

In Azure DevOps, click Pipelines and then click on your pipeline (contoso-university) to see your pipeline running.

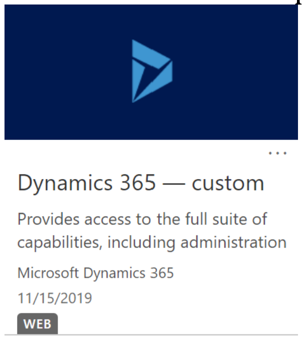
#20191115.1 Update pack-solution.yml for Azure Pipelines
Individual CI master 840566d
Just now
53s

It will take a few minutes for the import and publish to complete. Once it's done, proceed to the conclusion of this lab.

Conclusion

Let's verify that the solution has indeed been imported into our otherwise pristine production environment.

1. From the Power Platform admin portal, click Environments .
2. Click on the name of your production environment.
3. Click on the Environment URL.
4. Click on the custom application.


Dynamics 365 — custom
Provides access to the full suite of capabilities, including administration
Microsoft Dynamics 365
11/15/2019
WEB

5. Click the advanced find icon ( ) in the top-right corner.
6. In the window that opens, drop down the **Look for:** field and locate Departments. This is a custom entity from the managed solution we just imported and published.

Congratulations! You have now finished setting up the basic flow of taking code freshly committed to source control and pushing it directly into a production environment.