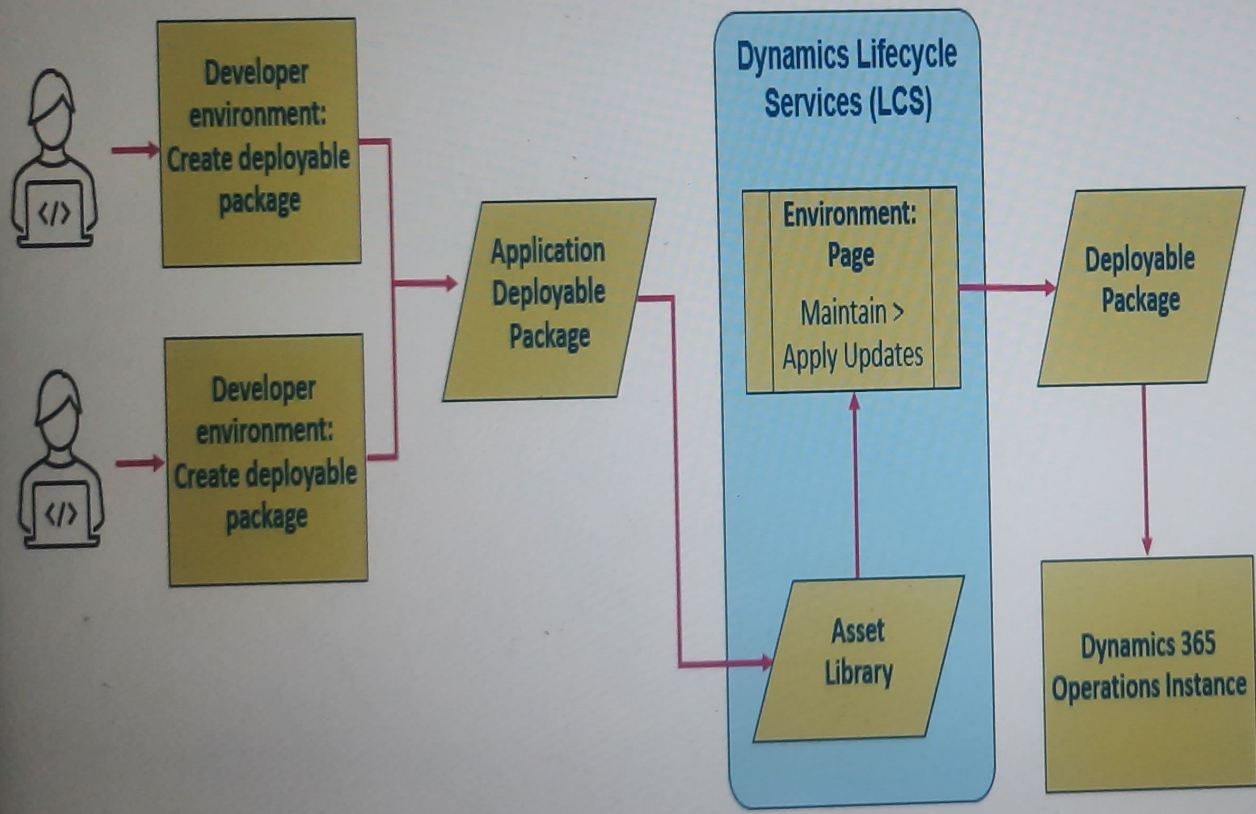
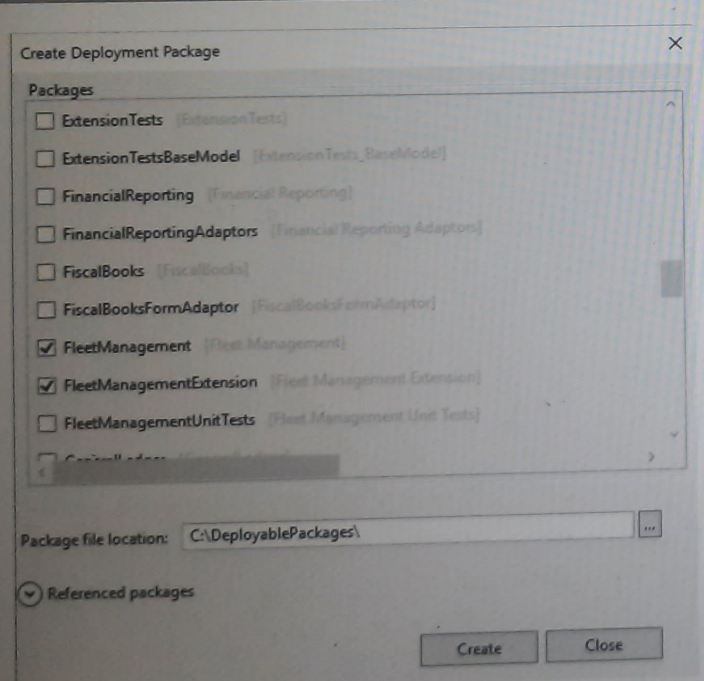
Release Process

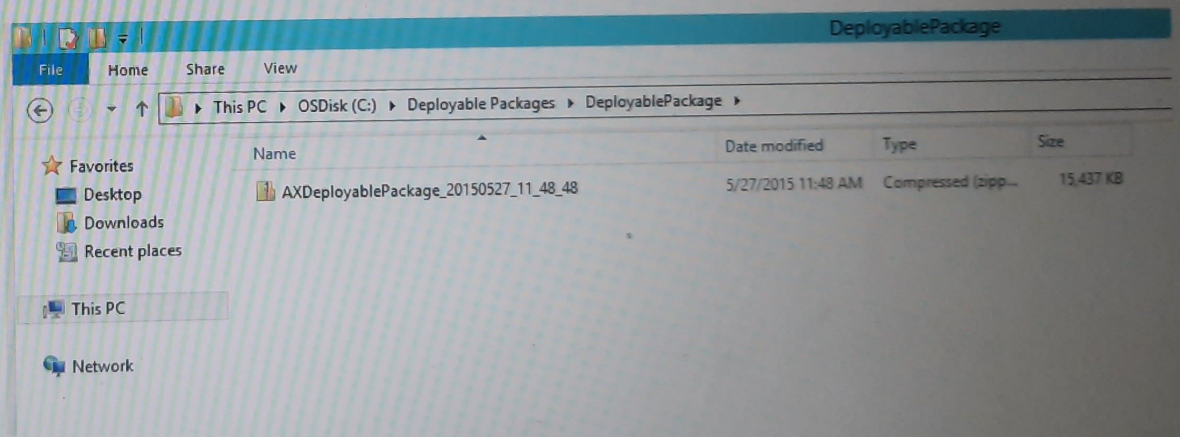


- To deploy code and customizations to a higher requirement a package needs to be created
- The packages created for deployment are known as Application Deployable Packages or AOT Deployable
- Create deployable packages using Visual Studio dev tools or build automation
- Upload package to LCS project's asset library
- Use LCS Maintain > Apply updates tool to apply **package** to runtime environment.

Deployment Process - Create a deployable package

- A deployable package shall be created in Development/Build environment. Build environment is recommended
- In Microsoft Visual Studio, select **Dynamics 365 > Deploy > Create Deployment Package.**
- Select the packages that contain your models, and then select a location in which to create the deployable package.





Deployment Process - Apply updates to cloud environments

Prerequisite steps

- Make sure that the package that should be applied is valid. There are three types of validations:
  - Basic package format validations
  - Platform version checks
  - Types of packages
- Make sure that the package(s) is applied in a sandbox environment before it's applied in the production environment.
- Make sure that the application binary update package is applied to your dev/build environment AFTER it is applied to your sandbox and production environment

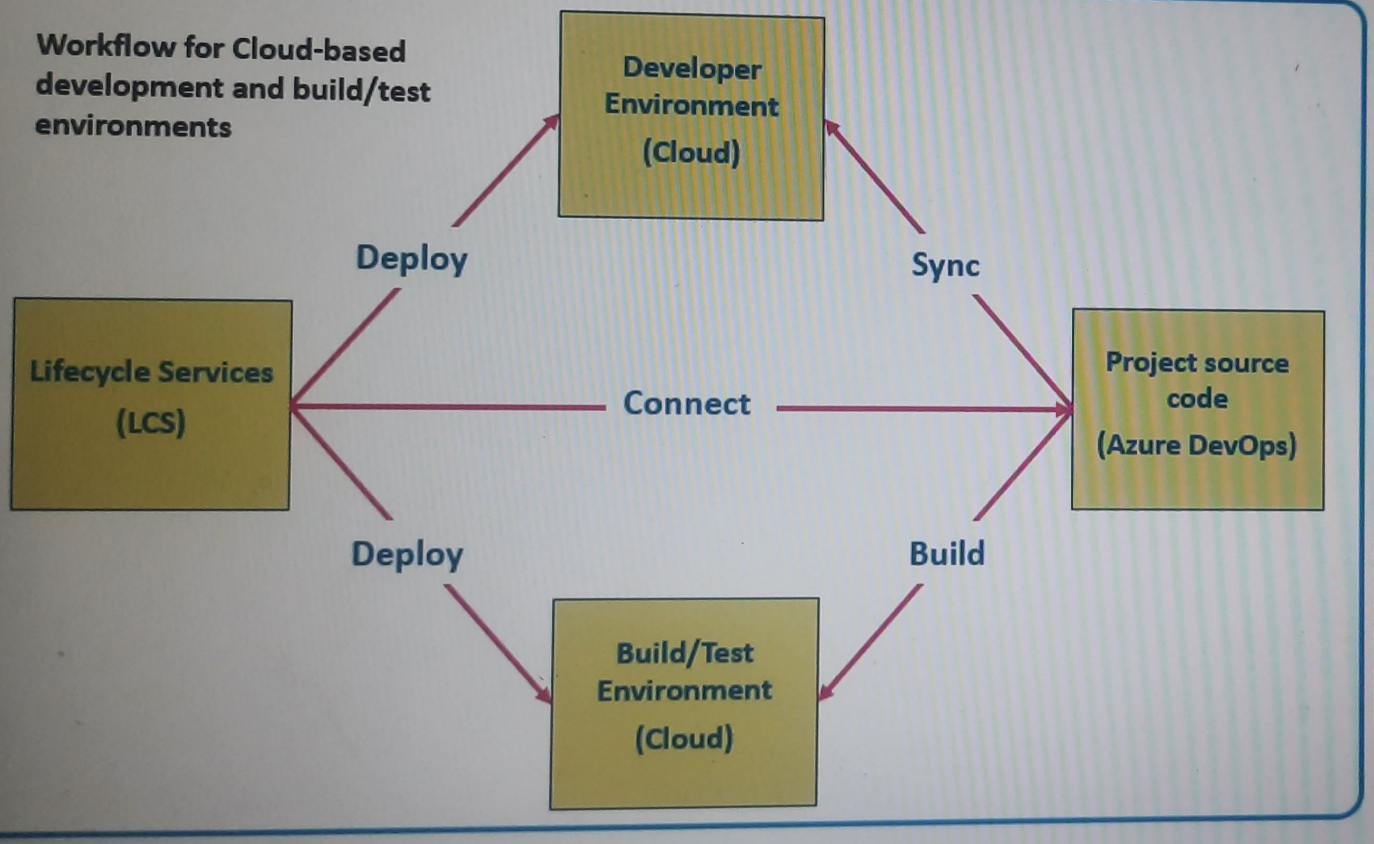
Supported package types

- AOT deployable package – Created by developer
- Application and Platform Binary update package – Released by Microsoft.
- Platform update package – Released by Microsoft.
- Commerce deployable package – A combination of various packages that are generated after the Commerce code is combined.
- Merged package – A package that is created by combining one package of each type.

Deployment Process - Apply updates to cloud environments

- **Non-Production Environment**
  1. Verify that the deployable package has been uploaded to the Asset library in LCS and Open the **Environment details** view
  2. Click **Maintain** > **Apply updates** to apply an update.
  3. Select the package to apply. Use the filter at the top to find your package.
  4. Click **Apply**. Check the status in the upper-right corner of the **Environment details** view changes from **Queued** to **In Progress**. a an **Environment updates** section now shows the progress of the package.
  5. When the package has been applied, the environment status changes to **Deployed**, and the servicing status changes to **Compl**
- **Production Environment**
  1. After the update is successfully applied in a sandbox environment, from the **Asset library** page, select the package from **Softwa deployable package** tab, and click **Release candidate**.
  2. Open the **Environment details** and Select **Maintain** > **Apply updates** to apply the package.
  3. Select the package to apply in your production environment, and then click **Schedule** to submit a request to apply it.
  4. Specify the date and time to schedule the package application. Click **Submit**, and then click **OK** to confirm.
  5. At the scheduled downtime, package deployment will start.
  6. The **Servicing status** field after the deployment is successfully completed, is set to **Completed**.

Deployment Process – Continuous Build and Test Automation

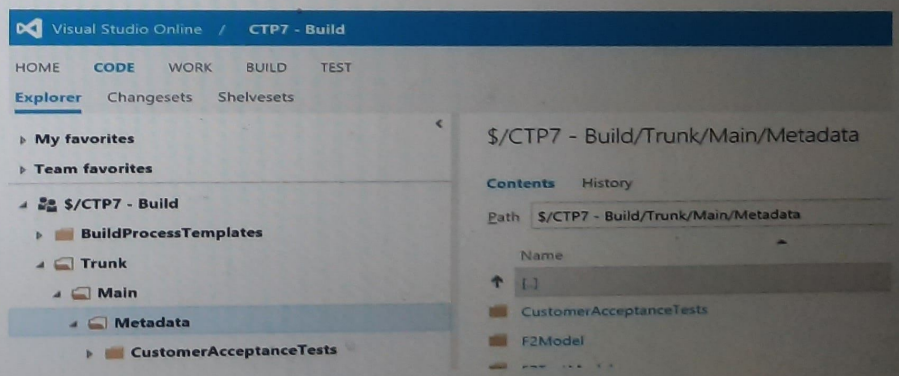


- Developer works on the source code on the developer VM, and the changes checked in the Azure DevOps project.
- The build process transfers the code from Azure DevOps to the build/test VM, generating deployable packages for use in sandbox and production environments.
- The source code is not directly transferred from the development VM to the build/test VM, it is instead synchronized through Azure DevOps.

Deployment Process – Continuous Build and Test Automation

- Set up Azure DevOps
  - Create a personal access token to authorize LCS access to Azure DevOps on their behalf.
  - Configure LCS
- Linking DevOps and LCS
  1. Login to the LCS portal to connect to Azure DevOps and your LCS project at https://lcs.dynamics.com/.
  2. Select a project that you are working on and Click the **Project Settings** tile.
  3. Select **Azure DevOps** and enter the Azure DevOps URL where the source code for your module project is located.
  4. Specify the Azure DevOps link, authorize, and then click **Choose default project**.

- Code check-in to Azure DevOps project.
  - All modules should be added to root folder **Metadata**.
  - Under each module, there should be two folders.
  - One for all models and other for descriptor XMLs.
  - Use LCS migration service for automatic check-in.



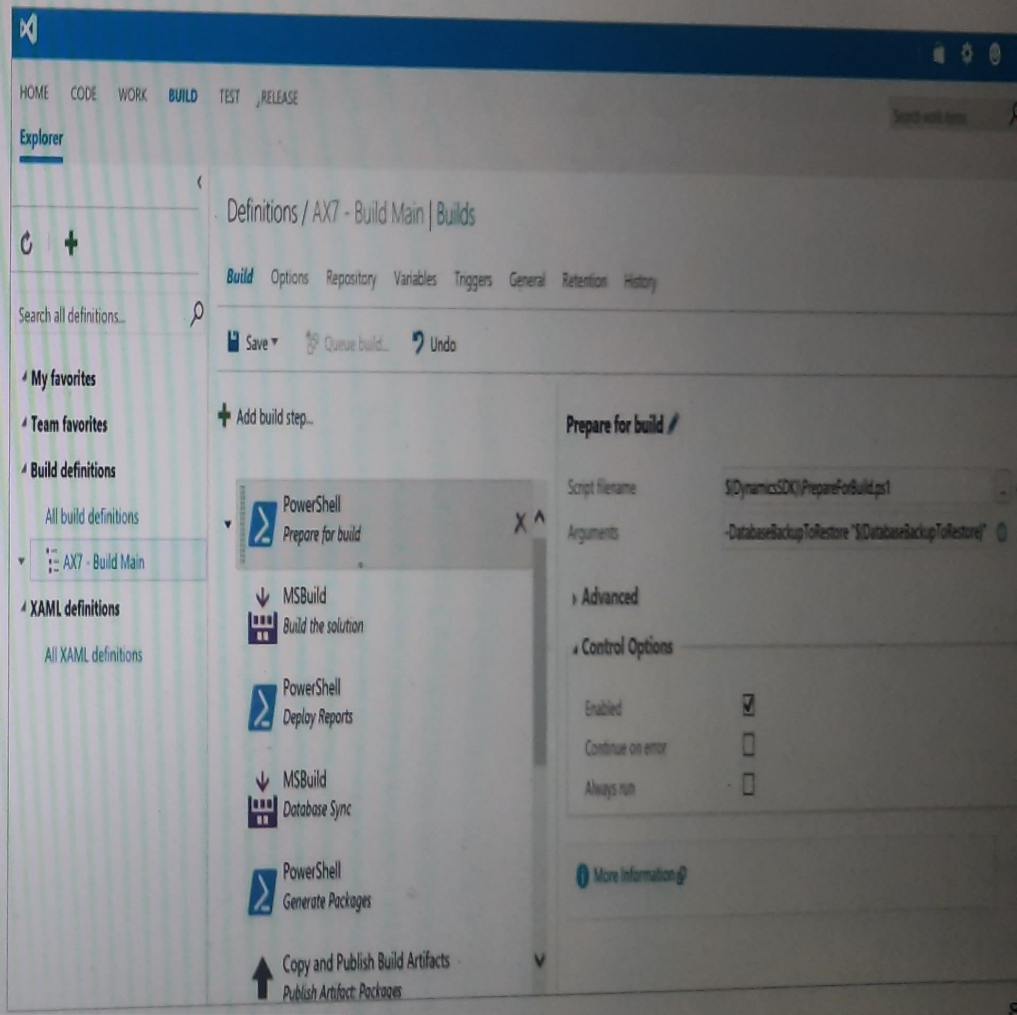
Deployment Process – Continuous Build and Test Automation

- Deploy build/test environments in Azure using LCS
  - In the deployment or configuration wizard, when prompted to **Select a Topology**, select **DevTest** then select a **Build and Test** topology.
  - You can configure the build agent name and build agent pool by clicking **Advanced settings**, select **Azure DevOps**
    - Build Agent Name - A name for build agent on Azure DevOps
    - Build Agent Pool - Specify build agent pool name which should be used for build machine deployment
    - Branch Name: Specify your Azure DevOps source code branch which will be default source code sync location for the buil VM. Default is "Main".
- Testing shall be integrated with the build either of the following
  - Using SysTest framework-based unit and component level tests
  - Using Task Recorder to generate XML code for automated test execution

Deployment Process – Continuous Build and Test Automation

- Build and verify the build and test execution results.

1. Configure the predefined variables parameters to be passed to the build.
2. Build the solution to build all modules under "Trunk/Main" branch
3. Use "Deploy Report" task to generate reports.
4. Use "Database Sync" task to sync the database to local SQL on build VM.
5. After successful build, create a deployable package that can be used to update sandbox/staging environment.
6. "Copy and publish build artifacts" uploads the deployable package to Azure DevOps artifacts location.
7. For test execution, there are three default tasks "Test Setup", "Execute Test" and "Test End“
8. The default build is scheduled to trigger every day at 5 P.M. It’s configurable.

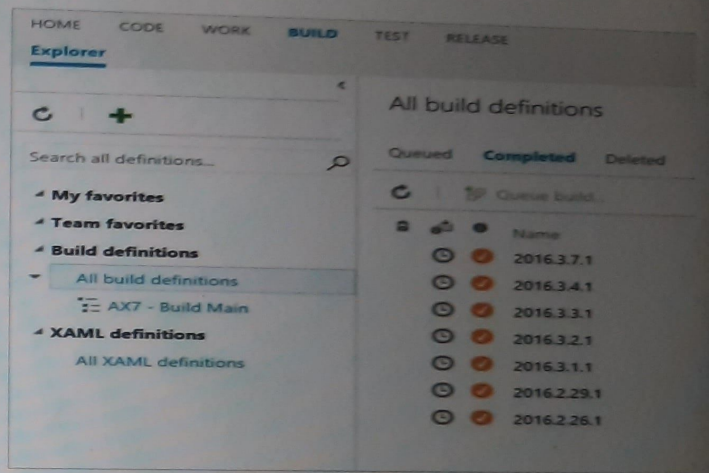


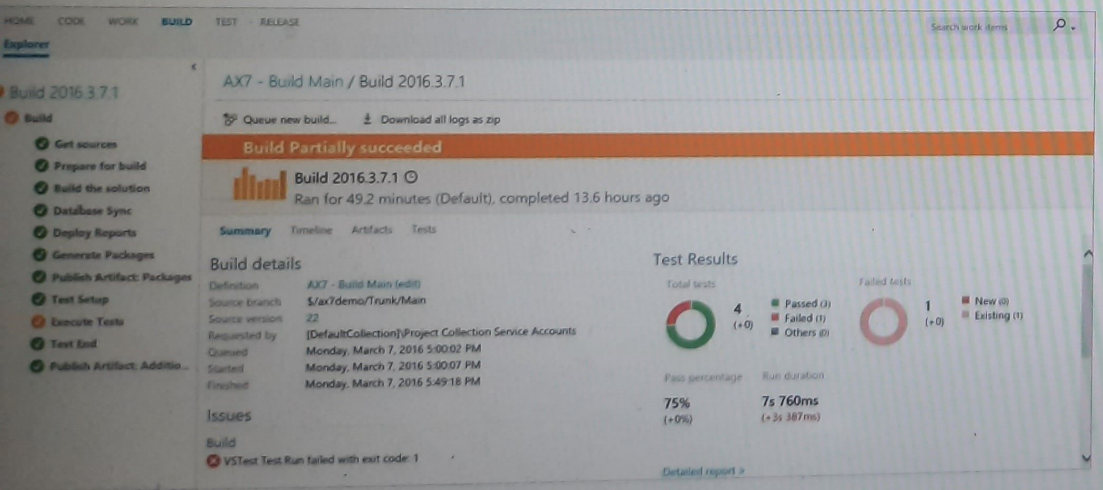
Deployment Process – Continuous Build and Test Automation

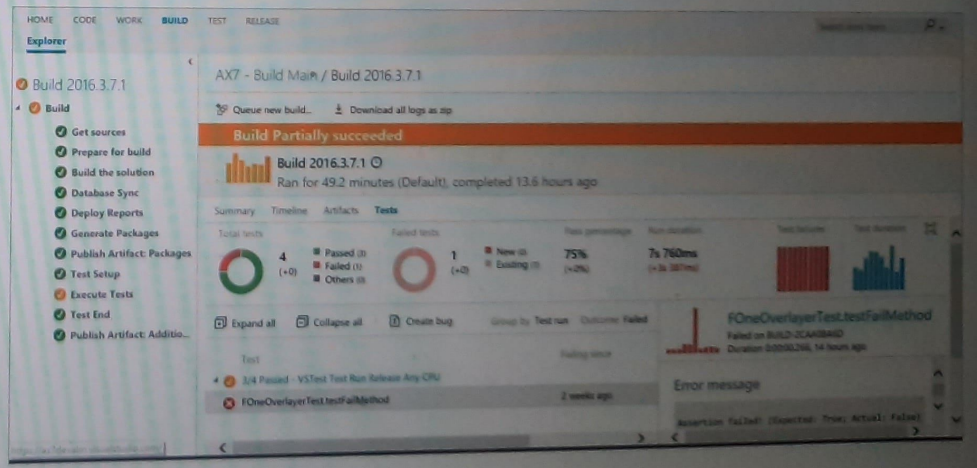
- Review and Configure Build VM environment. Manually trigger a build from Azure DevOps interface or VS IDE

1. Open your browser and connect to the Azure DevOps URL.
2. On the home page, under **Recent projects and solutions**, select a project.
3. From top links options, select **BUILD**.
4. On the left panel, select the default build definition instance.
5. Right-click and select **Queue Build** to trigger a build for your module and test module

Success or failure for the build will display, as shown in the examples. View all builds.







Deployment Process - Build Automation using Hosted agents and pipelines

- The X++ code build process can also be automated on any Microsoft Windows-based build agent.
- The cost of deploying build virtual machines, including setup and maintenance is avoided.
- Microsoft Build Engine (MSBuild) and custom X++ targets are utilized to build X++ code using .NET tools
- For this the Shared asset library in Microsoft Dynamics Lifecycle Services (LCS) provides NuGet packages that are required to be enables.
- For automating the build process the Azure DevOps also provides the below two types of Pipelines.
  - YML pipelines should be used for Git repositories
  - Classic pipelines are used for Team foundation version control (TFVC) repositories.
- A basic pipeline for compiling X++ code requires only installation of NuGet packages and build the solution
- A full pipeline for compiling X++ code should consist of the following steps
  - Install NuGet packages and Update the model versions
  - Build the solution or projects and install NuGet 3.3.0 agent
  - Create and publish the deployable package

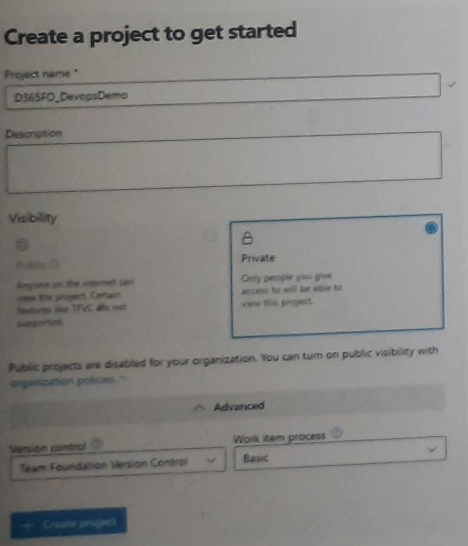
Source Control and Configuration Management

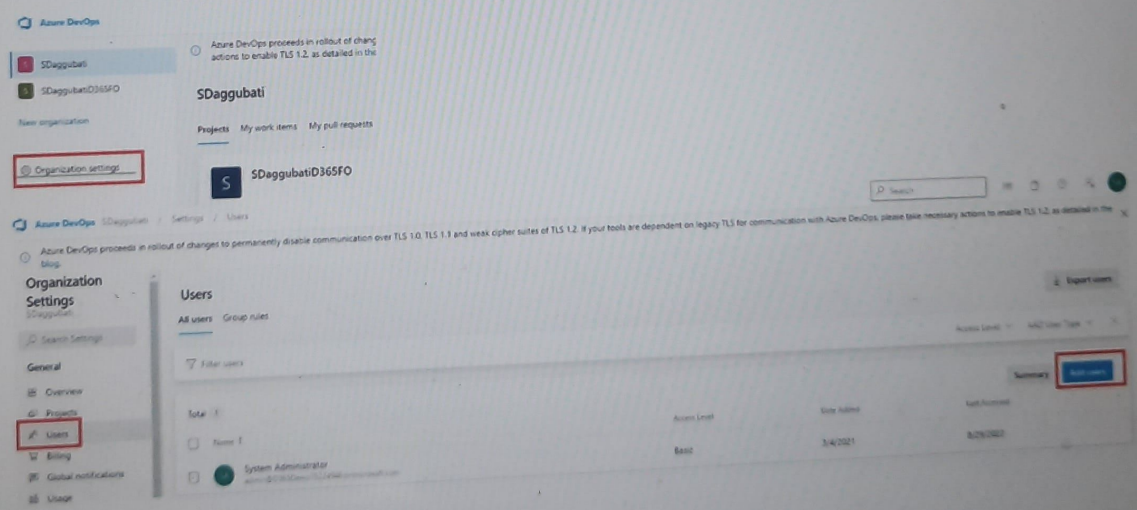
- Azure DevOps is used as the source control tool.
- Azure DevOps, offers an integrated set of services and tools to manage your software projects, from planning and development through testing and deployment.
- The following are the steps to configuring Microsoft Azure DevOps to enable source control on in a development VM.
    1. Configure your Azure DevOps organization and project
    2. Configure Visual Studio to connect to your team project
    3. Map your Azure DevOps project to your local model store and projects folder

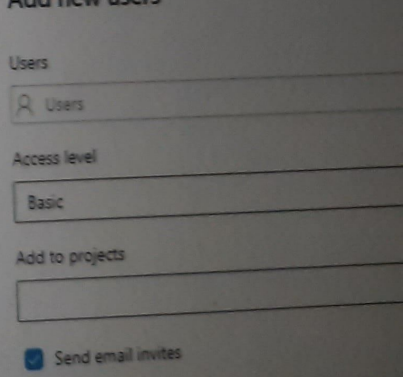
Source Control and Configuration Management

1. Configure your Azure DevOps Organization and Project - follow the below steps:

- Navigate to https://dev.azure.com/ to sign up for Azure DevOps. Click Start Free. Use your organizational account.
- Create Azure DevOps organization and select a URL for your account. You'll use this URL to connect when you're configuring source control in Visual Stu
- When the account is created, you're directed to your account main page where you're prompted to create your first project. Enter Project Name and description and select the version control of your choice.
- To add users to your DevOps project, click on Organization settings, Navigate to Users section, then click on the Add Users button.
- In the Add New Users dialog, Add the users in your organization that you deem necessary.





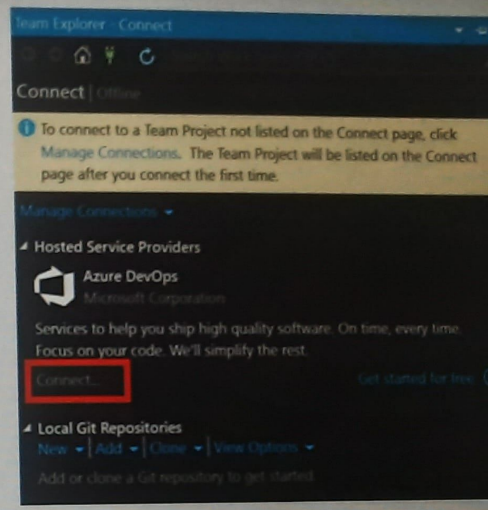


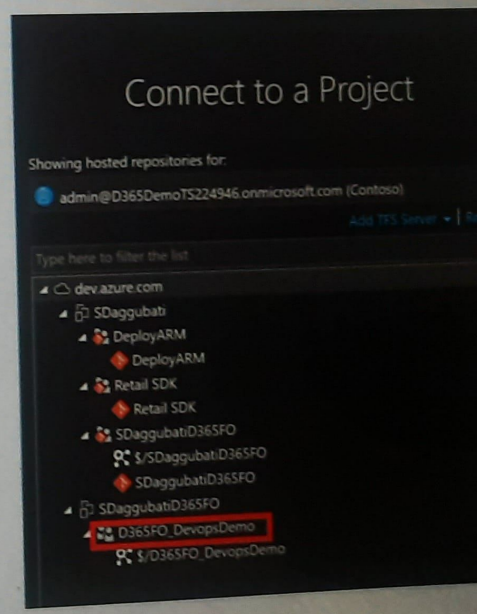
Source Control and Configuration Management

2. Configure Visual Studio to connect to Azure DevOps

To connect to an Azure DevOps project from Visual Studio, follow these steps:

1. Select **Manage Connections** in the **Team Explorer** window.
2. Right-click on the existing connections and select **Connect** to make the connection.
3. In the window that opens in the **Team Explorer** window, select **Manage Connections**.
4. Select **Connect to Team Project**. This will open a new dialog box.
5. In the dialog box, select **Servers**. This opens a new dialog box, where you will enter the Azure DevOps URL.
6. After you confirm these settings, a new dialog box where you will sign into Azure DevOps.
7. Select the box for the desired Azure DevOps project on the **Team Projects** pane.
8. Select **Connect**
9. Select **Advanced Configuration** to open a dialog box for adjusting the parameters of your workspace mapping.
10. Select **Advanced** in this dialog box for more options.
11. Change the workspace from a **Private** workspace to a **Public** workspace.
12. In the **Working folders** table, remove all existing mappings.



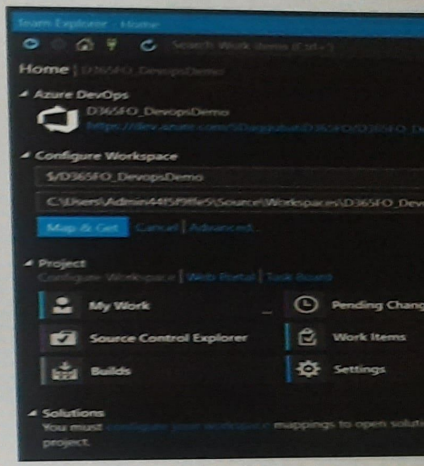


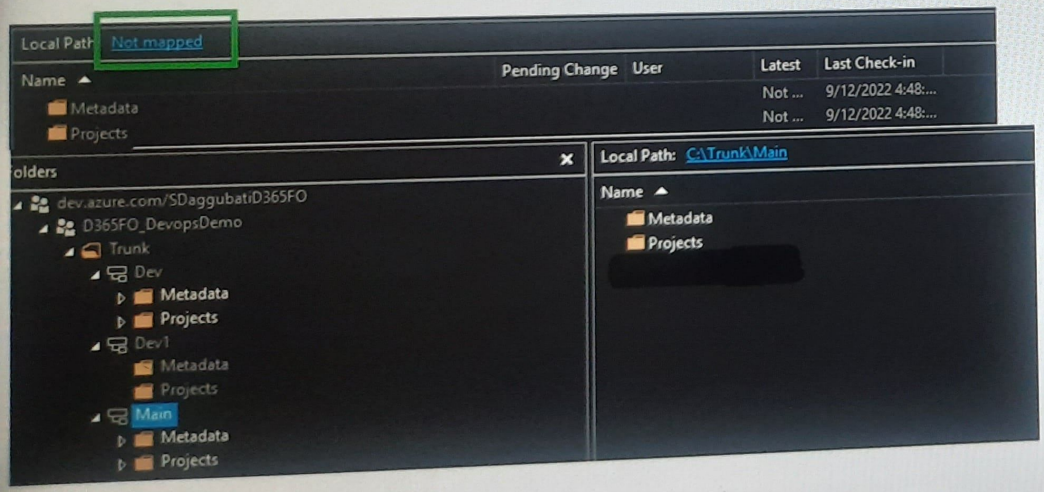
Source Control and Configuration Management

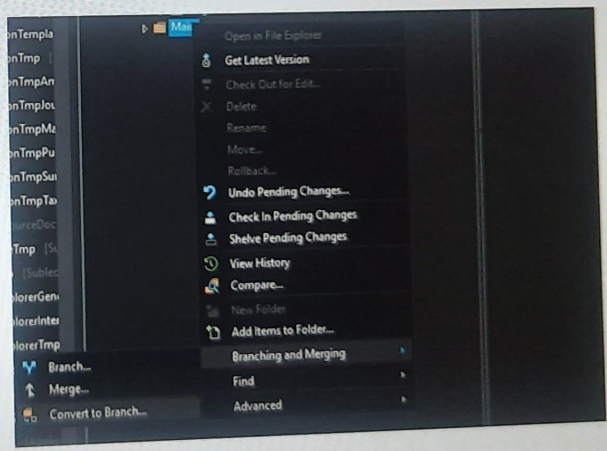
3. Map your Azure DevOps project to your local model store and projects folder

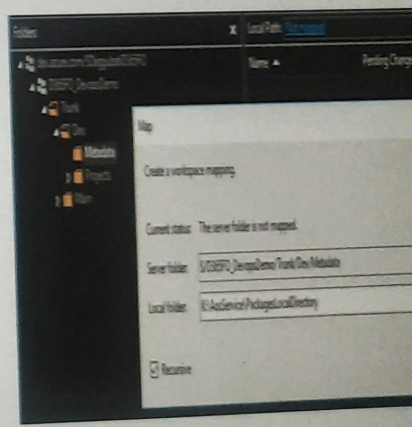
To connect to an Azure DevOps project from Visual Studio, follow these steps:

1. On a new row of the table, map the **Projects** folder of the desired branch of your Azure DevOps project to your local user's **Projects** folder for Visual Studio.
2. On another row in the table, map the **Metadata** folder of the same branch of your Azure DevOps project to the **Packages Local Directory** folder of the **AOS Service** drive.
3. Select **OK** and Select **Close** to exit all open dialog boxes.





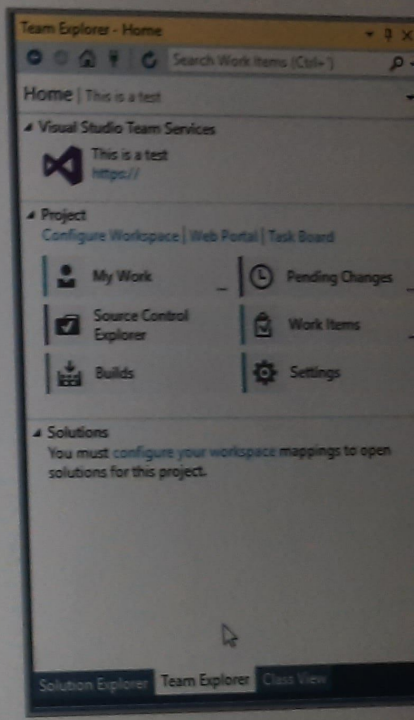




Source Control and Configuration Management

Configure Visual Studio to connect to Azure DevOps

To connect to an Azure DevOps project from Visual Studio, follow these steps:



1. Select **Manage Connections** in the **Team Explorer** window.
2. Right-click on the existing connections and select **Connect** to make the connection.
3. In the window that opens in the **Team Explorer** window, select **Manage Connections**.
4. Select **Connect to Team Project**. This will open a new dialog box.
5. In the dialog box, select **Servers**. This opens a new dialog box, where you will enter the Azure DevOps URL.
6. After you confirm these settings, a new dialog box where you will sign into Azure DevOps.
7. Select the box for the desired Azure DevOps project on the **Team Projects** pane.
8. Select **Connect**
9. Select **Advanced Configuration** to open a dialog box for adjusting the parameters of your workspace mapping.
10. Select **Advanced** in this dialog box for more options.
11. Change the workspace from a **Private** workspace to a **Public** workspace.
12. In the **Working folders** table, remove all existing mappings.
13. On a new row of the table, map the **Projects** folder of the desired branch of your Azure DevOps project to your local user's **Projects** folder for Visual S
14. On another row in the table, map the **Metadata** folder of the same branch of your Azure DevOps project to the **Packages Local Directory** folder of the **Service** drive.
15. Select **OK** and Select **Close** to exit all open dialog boxes.