CI/CD Using Azure Pipelines

Accentient™

Azure DevOps | 2022.2

ACCENTIENT EDUCATION SERIES

Committed to training success

www.accentient.com

CI/CD Using Azure Pipelines

Course Number:	**CICD**
Version:	**2022.2**
Software version:	**ADSvc**

Disclaimer

While Accentient takes great care to ensure the accuracy and quality of these materials, all material is provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

CI/CD Using Azure Pipelines

Course Introduction

"We need to figure out a way to deliver software so fast that our customers don't have time to change their minds."

- Mary Poppendieck

Accentient

Accentient

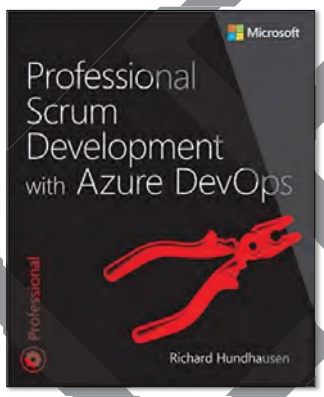
- A leader in ALM, DevOps, and Scrum knowledge
- Helped thousands of teams and individuals understand and implement Azure DevOps/VSTS/TFS and Scrum successfully
- Has a close working relationship with Microsoft
- Course creator and steward for Scrum.org
- Has trainers that are Microsoft MVPs, Professional Scrum Developers, Professional Scrum Trainers, and authors

www.accentient.com | @accentient

Accentient

Course Creator: Richard Hundhausen

- President of Accentient
- Author of software development books
- First Microsoft TFS/ALM/DevOps MVP
- Professional Scrum Developer
- Professional Scrum Trainer
- Co-creator of Nexus scaled Scrum Fx
- richard@accentient.com
- @rhundhausen



http://bit.ly/PSDAzDo

PSD PST

Accentient

Prerequisites

- Ideally, you …
  - Work on a team
  - Develop and deliver a software product
  - Use Microsoft tools and technologies
  - Use Azure DevOps
  - Automate your build and release processes
  - Want to shorten your feedback loops

Accentient

Introductions

- Name, company, title/role
- Software development experience
- Azure DevOps experience
- Azure Pipelines experience
- Expectations

Accentient

Course Overview

This course demonstrates how to use Azure Pipelines to practiced Continuous Integration (CI), Continuous Delivery (CD), and Continuous Deployment (CD)

- Introduction to Azure Pipelines
- Continuous Integration with Azure Pipelines
- Continuous Delivery with Azure Pipelines
- Continuous Deployment

Accentient

Our Shared Development Environment

- We will be using a shared instance of Azure DevOps
- We will be working in teams and each team will …
  - Be collocated (physically or virtually)
  - Have its own Azure DevOps project
  - Collaborate on all work in this class
- Each team member will …
  - Need a Microsoft Account

Accentient

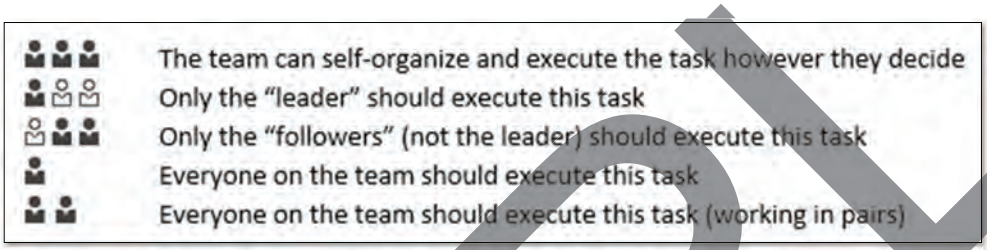
Schedule and Logistics

- Breaks
  - When should we have breaks?
- Labs
  - Labs can be breaks too
- Lunch
  - When should we break for lunch?

Accentient

Collaborating as a Team

- There are many opportunities for collaboration in this course
  - Some tasks, however, must be performed by *one* team member
- All tasks will be marked with an appropriate icon …



Accentient

CI/CD Using Azure Pipelines

Module 1
Continuous Integration

"If it hurts, do it more frequently, and bring the pain forward."
- Jez Humble

Accentient

Module Backlog

What are we forecasting for this module?

- Azure Pipelines
- Build Pipelines
  - Automated Testing
- Continuous Integration
  - Team practices
  - Test Impact Analysis
  - Continuous Integration+
- Lab

Accentient

Azure Pipelines

- An Azure DevOps Service for building, testing, and deploying
  - Provides continuous integration (CI) and continuous delivery (CD) services
- From any version control system …
  - Azure Repos, GitHub, Bitbucket, SVN
- For any language …
  - Node.js, Python, Java, PHP, Ruby, Go, C/C++, .NET, Android, iOS, …
- To any platform …
  - Linux, macOS, Windows
  - On-premises, cloud, container

Visit http://bit.ly/3100sXF for more information

Accentient

Build Pipelines

Accentient

Build Pipeline

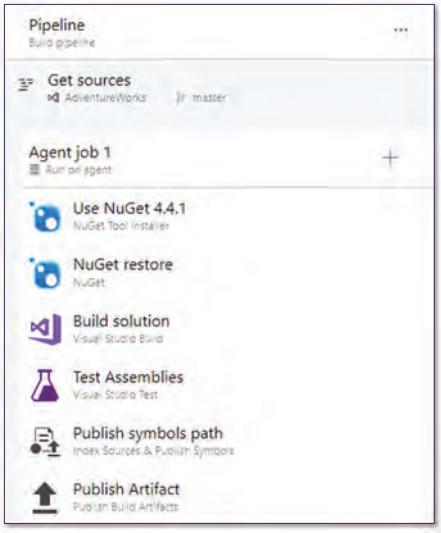
- A pipeline for automating the build of your application
- Build pipelines enable you to …
  - Create, manage, and run build scripts
  - Automate the compiling and testing of simple to complex apps

  - Run other utility, package, deployment, or tool installation tasks

Accentient

Build Pipelines Automate Everything

- Automates all kinds of tasks …
  - Getting the code
  - Installing NuGet (and other tools)
  - Restoring NuGet and other packages
  - Building
  - Testing
  - Packaging
  - Deploying
  - Any other scripts and utilities



Visit https://bit.ly/36DXx81 for more information

Accentient

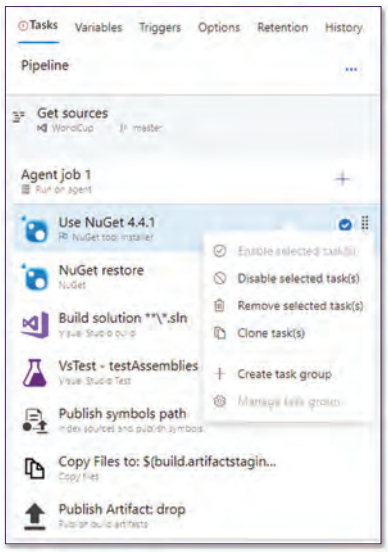
Creating a Build Pipeline

- Use classic/visual designer or configure as code (YAML)
  - Select the source, project, repository
  - Select and apply a template
- When defining a new build, specify …
  - Pipeline settings (name, agent pool, parameters)
  - Source settings
  - Agent job settings (pool, demands, plan, parallelism)
  - Task settings (specific to the type of task)
  - Variables
  - Triggers
  - Retention policy
  - Options (build number format, work items, badges)

Accentient

Tasks

- Tasks are the building blocks for defining automation in a pipeline
  - A task is simply a packaged script or procedure that has been abstracted with a set of inputs
- Tasks can be added, disabled, removed, reordered, or cloned

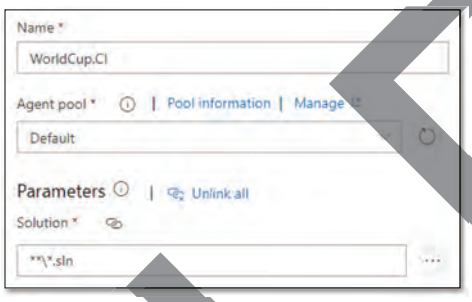


Visit http://bit.ly/2w26xnm for more information

Accentient

Pipeline Settings

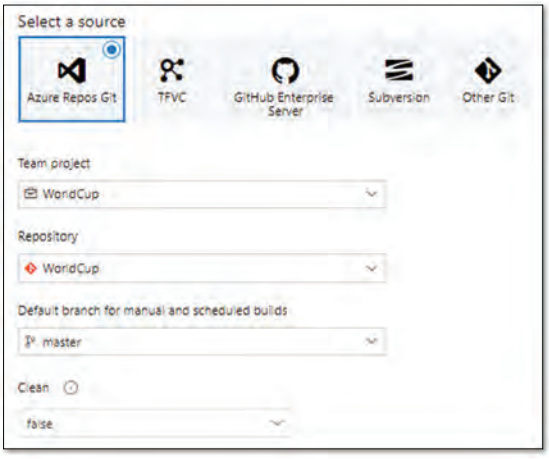
- Miscellaneous settings
  - Pipeline name
  - Agent pool
- Parameters
  - Allow you to centrally manage important parameters (e.g. Visual Studio solution) for tasks used across the entire pipeline
  - Build templates include pre-defined process parameters
  - You can link/unlink fields to/from parameters



Accentient

Get Sources

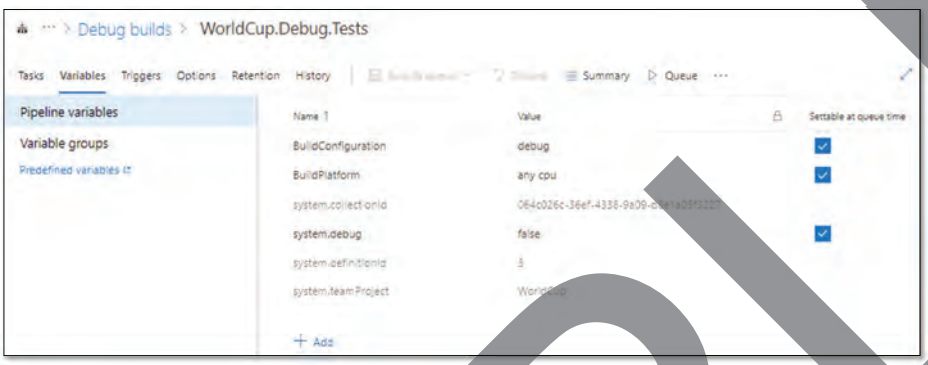
- Build can fetch code from many sources
  - Current project (Git or TFVC)
  - GitHub
  - GitHub Enterprise Server
  - Bitbucket Cloud
  - Subversion
  - Other Git



Accentient

Variables

- Create and use variables to pass information to build tasks



  - Many predefined variables exist (http://bit.ly/2nJYLb3)

Accentient

Triggers

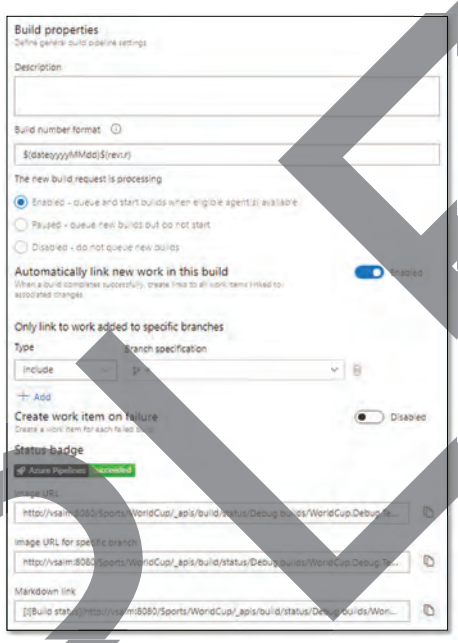
- Builds can be queued
  - Manually
  - Automatically (triggers)
- Triggers can fire …
  - When a push/check-in occurs (a.k.a. Continuous Integration)
  - When a pull request is made (a.k.a. PR trigger)
  - At a scheduled time
  - When another build completes (a.k.a. build chaining)
- Triggered builds can also be queued manually

Visit http://bit.ly/2WrBnyr for more information

Accentient

Options - Build Properties

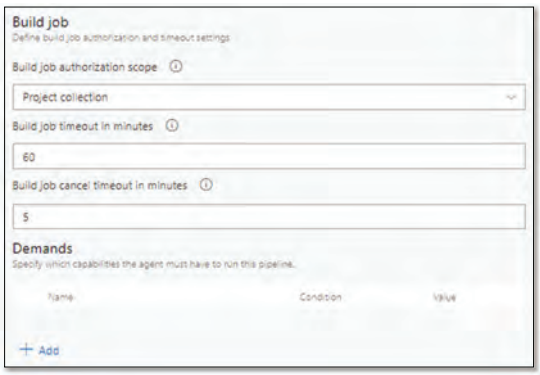
- Description
- Number format
  - Use a combination of tokens, variables, and underscore characters to generate more meaningful build names
- New build request processing
  - Options for queueing and starting new builds
- Work item linking and creating
  - Link or create associated work items
- Badge enabled
  - Show the latest outcome on external web sites



Accentient

Options - Build Job

- Job authorization scope
  - *Project Collection* provides access to multiple projects; *Current Project* doesn't
- Job timeout
  - Maximum time a build job is allowed to run on an agent before being canceled
- Job cancel timeout
  - Maximum time a build job is allowed to respond after the user cancels the build
- Demands
  - Ensure the capabilities your build needs are present on the build agents

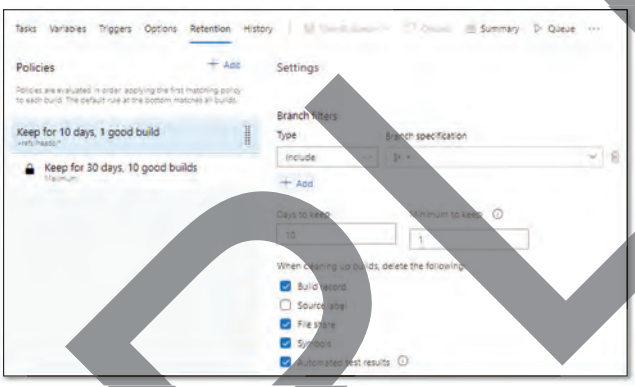


Visit https://bit.ly/2Mr0nod for more information

Accentient

Retention

- Retention policies are used to configure how long builds are to be retained by the system (to conserve storage and reduce clutter)
  - You can configure the number of days to keep as well as what to delete
  - Policies can be added and removed
  - Policies are evaluated in the order listed
- Specific builds can be retained forever
  - For auditing/tracking

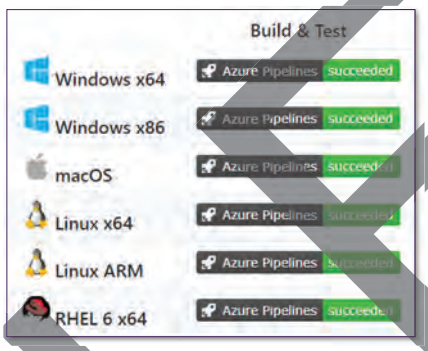


Accentient

Agents

Accentient

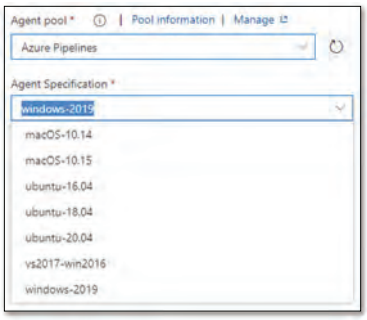
Build and Release Agents



- To build (or release) your code, you need at least one agent
  - An agent is installable software that runs one build job at a time
- Microsoft-hosted or self-hosted
  - Hosted agents are maintained by Microsoft
  - Private agents can be installed on Windows, Linux, or OSX
- Parallel jobs
  - As you add more code and people, you might need to use multiple Microsoft-hosted or self-hosted agents at the same time

Accentient

Microsoft-Hosted Agents



- Azure Pipelines offer several Microsoft-hosted agent pools
  - Microsoft handles maintenance/upgrades
  - Private projects have one free parallel job with limited minutes
  - Public projects have free parallel jobs
- Microsoft-hosted agents are deployed with specific software
  - Visit http://bit.ly/2QClySq for the exact list

Visit http://bit.ly/2qVxx2p for more information

Accentient

Windows Agents

- To build Windows, Azure, and other Visual Studio solutions
  - Windows agents can also build Java and Android apps
  - Written for .NET Core in C#
- Build agent machine requires
  - PowerShell 3 or newer
  - Visual Studio 2015 or later is recommended
- Deploy and run the agent
  - Download the agent from your browser
  - Must be an *Agent Pool Administrator*
- Can be run as a service or interactively (for UI tests)

Accentient

YAML

Accentient

YAML Pipelines

- YAML is the language of Azure Pipelines
- YAML allows pipelines to be configured as code
  - Example: To update a build pipeline, you simply update the code
- YAML provides the advantages of configuration as code
  - The pipeline is stored and versioned alongside your code
  - Changes to the pipeline can be reviewed/validated alongside application code
  - If a change to the pipeline causes a break or results in an unexpected outcome, you can more easily identify the issue

Visit https://aka.ms/yaml for more information

Accentient

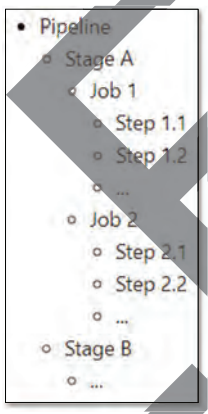
Creating a YAML Pipeline

- Creating a YAML pipeline is straightforward
  1. Select the repository
  2. Select a template, starter pipeline, or existing YAML file
  3. Customize the YAML file (using the assistant if you'd like)
  4. Save the YAML file to the repository (default name is *azure-pipelines.yml*)
- The pipeline can be run manually or triggered in a number of ways

Accentient

Anatomy of a YAML Pipeline

- A pipeline contains one or more stages that describe a build or release process
  - Stages are the major divisions in a pipeline (e.g. "build app", "run tests", "deploy to prod")
- A stage contains one or more jobs
  - Jobs are the units of work that run on the same machine
  - Execution can be configured based on dependencies
- A job contains a linear series of steps
  - Steps can be tasks, scripts, or references to external templates.



Accentient

Automated Testing

Accentient

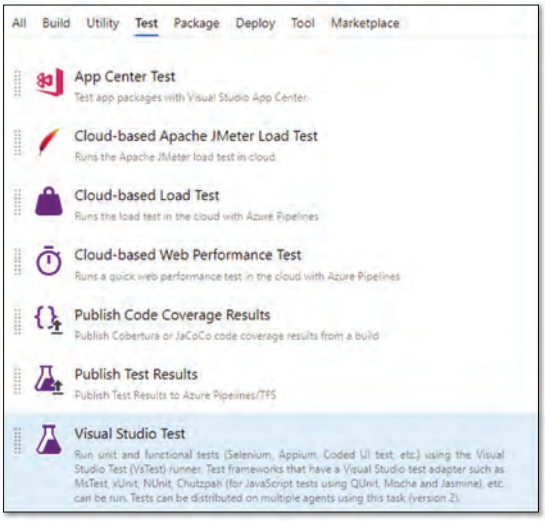
Automated Regression Testing

- After changing any code, you should build and run tests
  - You do this by re-running all applicable tests to determine whether the changes broke anything that worked prior to the change
- Automated builds running verification tests is a great way of automating regression testing

Accentient

Running Tests in the Pipeline

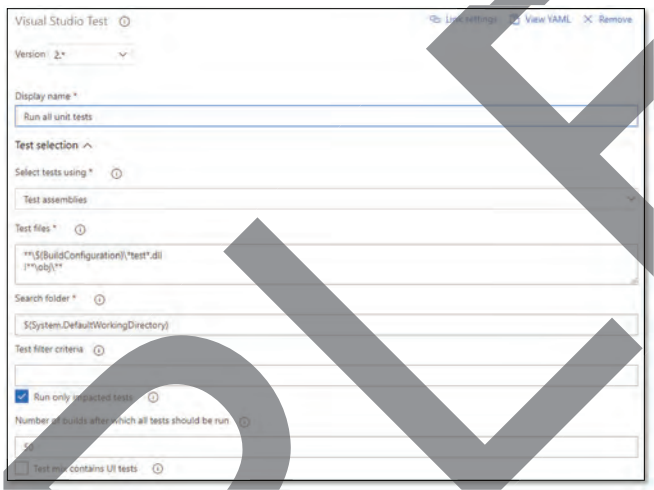
- As part of your pipeline, you can add one or more test tasks
  - Before running tests, you should confirm that the project contains tests, that they pass, and that they are also stored in the repository



Accentient

The Visual Studio Test Task

- Run tests in the pipeline with the Visual Studio Test task
  - Unit and integration
  - Functional (Selenium, Appium)
  - Other frameworks with a Visual Studio adapter (xUnit, NUnit, Chutzpah, etc.)
- Test Impact Analysis (TIA) capability can be enabled



Visit http://bit.ly/2VLapEH for more information

Accentient

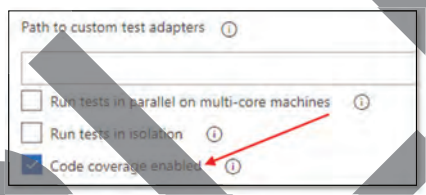
Code Coverage

- Code coverage lets you analyze how much of your code was exercised during the execution of a set of unit tests
  - Results are available as a percentage, as well as graphically
  - Applies to both managed and unmanaged (native) code
- Code Coverage is a way to measure the effectiveness of your unit tests
  - On a block-by-block or even line-by-line basis
- Requires Visual Studio Enterprise edition
  - *DotCover*, *OpenCover*, and *NCover* are 3rd party alternatives

Accentient

Gathering Code Coverage Data

- Code coverage data can be gathered during a build
  - Enable Code coverage on the Visual Studio Test task
- Microsoft-hosted agents support this
  - Visual Studio Enterprise edition must be installed on self-hosted agents



Accentient

Using Code Coverage as a Metric

- Don't use Code Coverage as a "feel-good" quality metric
  - While low numbers will tell you that you need more tests, higher numbers don't necessarily mean that you've written good tests
  - Just because you're fully covered doesn't mean you're fully tested
- Use the code coverage metric wisely
  - Example: Your team should take interest in the overall code coverage trend – increasing (or at least not decreasing) over time/per commit/per build

Accentient

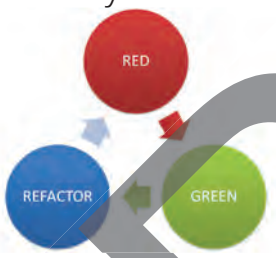
Continuous Integration

Faster and Faster Feedback Loops

- One of your team's goals should be to create faster and faster feedback loops
  - Automated testing, building, and integration can help achieve this
- The team can be informed when a change is introduced that alters the functioning and deployable state of the software
  - Developers can fix the problems sooner before switching context
- Quality is increased while waste is reduced

Continuous Integration (CI)

- A development practice where developers integrate (check-in) their code changes regularly
  - Integration becomes a part of the natural rhythm of coding
  - Enables test-code-refactor and test-first development
  - The team can progress steadily forward by taking small steps

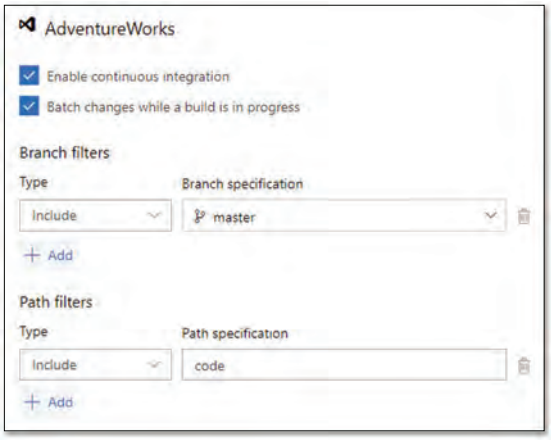


Visit https://thght.works/1khpAOV for more information

Accentient

Continuous Integration (CI) Pipelines

- Use the Continuous Integration (CI) build pipeline trigger
- *Batch changes* option
  - Ensures that only one such build is running at any time
  - Any pushes/check-ins that occur while a build is running are combined and built together when the current build completes



Visit http://bit.ly/2WrBnyr for more information

Accentient

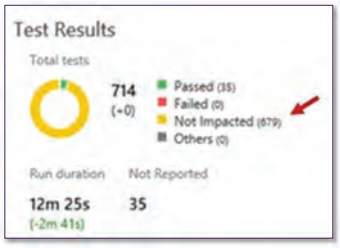
Practice Continuous Integration

- Adopt a "collective ownership" attitude within the team
  - Integrate regularly, no long-lived branches, swarming
- Configure Azure Pipelines to build and test on check-in
  - Optimize the build process for fast feedback (e.g. < 60 seconds)
  - Test in a clone of the production environment
- Be transparent
  - Fix problems as fast as you find them

Accentient

Test Impact Analysis

- What if you have too many tests to run them all every time?
  - Assuming these are valuable tests, then this is a good problem to have!
- Test Impact Analysis (TIA)
  - A technique/feature that determines the subset of tests for a given set of changes
  - In other words, for a given code commit entering the pipeline Test Impact Analysis will select and run only the relevant tests required to validate that commit

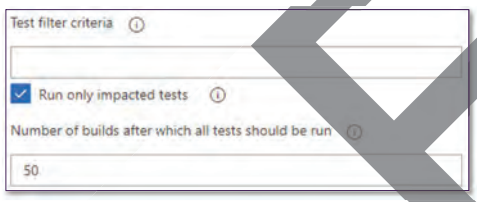


Visit http://bit.ly/2qLHCid for more information

Accentient

Configuring Test Impact Analysis

- Check the box and the Test Impact data collector is automatically configured
  - The *Number of builds after which all tests should be run* setting will automatically run all tests after that many builds, to rebuild the mapping between tests and source code on a regular basis



- Note: If your application interacts with a service in the context of IIS, you must also configure the Test Impact data collector to run in the context of IIS by using a *.runsettings* file
  - Visit http://bit.ly/2VAhrJ1 for more information

Accentient

Test Impact Analysis: How Tests Get Selected

- Test Impact Analysis will look at an incoming commit, and select the set of relevant tests …
  - Existing tests impacted by the incoming commit
  - Plus any previously failing tests (TIA tracks these)
  - Plus any newly added tests

- Note: As of this writing, Test Impact Analysis does not yet support .NET Core, .NET 5.0, or .NET 6.0

Accentient

Continuous Integration+

Continuous Integration +

- Continuous Integration typically refers to
  - Integrating code from one or more developers/branches
  - Ensuring that the integrated code passes unit tests
- Continuous Delivery/DevOps, mandates that we also
  - Run on production-like environments
  - Passing acceptance tests

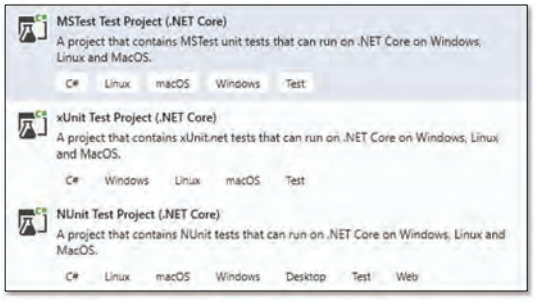
Acceptance Tests

- Verifies that the team is *building the right thing*
- Created and run by anyone on the development team
  - Traditionally a responsibility of the Quality Assurance (QA) or Business Analyst (BA) role
- Can be automated or manual
  - For Continuous Delivery, they must be automated
- There are many types of acceptance tests:
  - Functional tests
  - Performance tests
  - Load tests
  - Security tests

Accentient

Acceptance Testing Using Unit Testing Frameworks

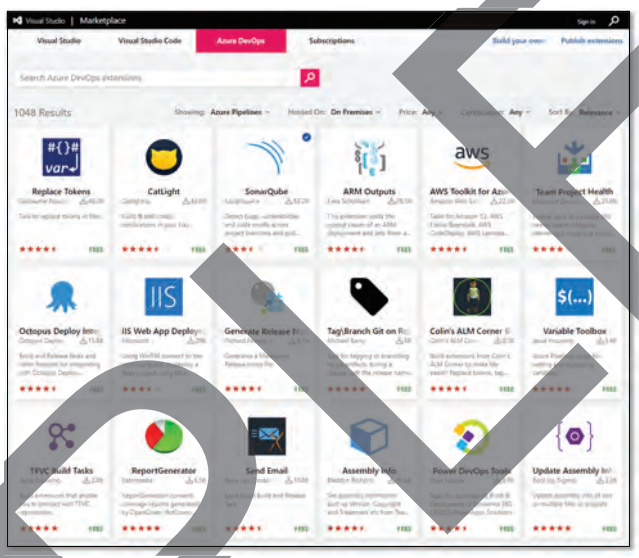
- Any of the unit testing projects and unit tests can be used for automated acceptance testing
- Unit testing frameworks aren't just for writing and running your classic unit tests
  - You can write and run awesome acceptance tests too
- Note: For C++ acceptance tests, you can install and use Boost.Test or Google Test



Accentient

Azure Pipelines Extensions in the Marketplace

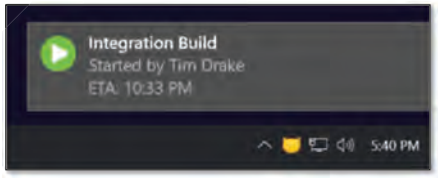
- The Azure DevOps Marketplace has many Azure Pipelines extensions
  - Created by Microsoft and partners; many are free
  - http://bit.ly/2ordS73



Accentient

CatLight

- A notification app for developers
- CatLight shows the current status of continuous delivery and informs when attention is needed
  - See the status of the things you track in the tray



  - See more details on the dashboard

Visit https://catlight.io for more information

Accentient

Module Review

What have we accomplished and learned?

- Continuous Integration = automated building and testing
  - But also the practice of team members integrating their code often
- Don't be afraid to push changes frequently
  - Broken builds and failed tests are a type of feedback
  - The more often you integrate (push), the less pain it will be
- Code coverage tells you where to invest in more testing
- Test Impact Analysis and test filters further shorten the feedback loop
- Put everything into version control, not just the code
  - Tests, scripts, documentation, configuration, tools, SDKs, etc.
  - YAML pipelines are an example of this

Accentient

Lab

In this lab you will create and configure build pipelines to include running automated tests in various ways.

- Create and use a build pipeline
- Include automated tests in the build pipeline
- Enable and practice Continuous Integration
- Use YAML to Configure a Build Pipeline (optional)

Accentient

Lab 1: Practicing Continuous Integration

CI/CD Using Azure Pipelines

LAB OVERVIEW

This lab walks you and your colleagues through the process of forming into teams, provisioning the Azure DevOps Services organization and project, importing the codebase, and creating and running build pipelines in various ways.

Estimated time to complete this lab: **60 minutes**

Task Execution

As this is a team-based training course, there are a number of opportunities for team members to learn to collaborate more effectively. Unfortunately, there is a possibility for team members to accidentally impede, block, or otherwise cause unintentional conflicts. To minimize the possibility of conflicts, critical tasks in this course have been marked with an icon indicating who on the team should execute the task:

-  The team can self-organize and execute the task however they decide
-  Only the "leader" should execute this task
-  Only the "followers" (not the leader) should execute this task
-  Everyone on the team should execute this task
-  Everyone on the team should execute this task (working in pairs)

Tip: Look for the "leader"  tasks and ensure that they are only performed once per team. Also, ensure that the "follower"  tasks are only performed by everyone else (not the leader).

Teams of One

If you are working by yourself and not on a team, make sure to perform all of the "leader"  tasks, and none of the "follower"  tasks. This scenario is common for students learning remotely.

EXERCISE 1 – SET UP THE ENVIRONMENT

Task: Install Courseware Files

In this task you will install the files required by this class.

Dependencies

- Signed in as *Administrator*

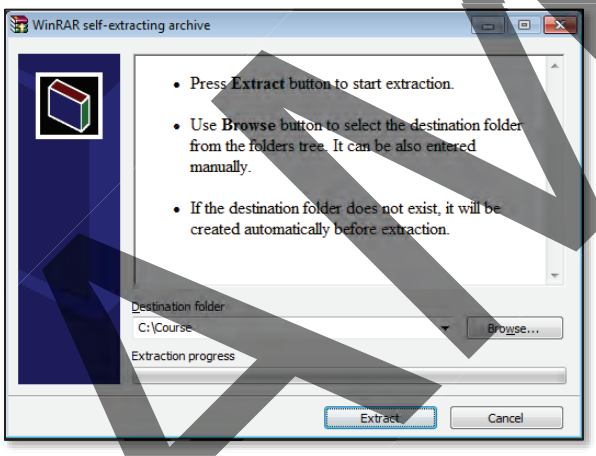
1. Verify that the **C:\Course** folder does not exist.

   Note: If this folder already exists, then your computer may have been used for a prior training class. If that is the case, then the effectiveness of the hands-on labs that follow may be diminished.

2. If necessary, copy the courseware file to your desktop.

   This file may already be on the desktop. If not, you may have to ask your instructor for help locating and/or copying this file. If you cannot locate this file, please email support@accentient.com to obtain a copy.

3. Extract the courseware files, specifying **C:\Course** as the **Destination folder**.



   It can take a few moments to extract the files. The folder C:\Course will be created during the process. After extracting the files, you should have one or more of the following sub-folders:

- C:\Course\Guidance
- C:\Course\Labs
- C:\Course\Software

Task: Form Into Teams (optional)

If necessary, your instructor will facilitate the creation of equally sized (5 team members or less), cross-functional, collocated teams.

- Form into cross-functional teams (of 5 members or less)
- Do your best to ensure experts aren't all on the same team …
  - Git experts
  - Visual Studio experts
  - Azure DevOps/VSTS/TFS experts
- Collocate
- Introduce yourself (if necessary)
- Decided on a team name (i.e. "Team Blue", "The Honey Badgers", "Repo Team", etc.)
- Write your name and team name where it is visible to others in the class

What is the name of your team? _____

Who are your team members? _____

_____

Task: Identify a Microsoft Account

In this task you will identify, or create if necessary, a Microsoft Account that you will use for the various online services leveraged during this class.

What is your Microsoft Account? _____

1. If you already have a Microsoft Account, write it on the line above and skip the rest of these steps.
2. Open **Chrome** and navigate to https://signup.live.com.

   If you don't have Google Chrome installed, you can install it from here: www.google.com/chrome.
3. Provide the required information.

   This will include your name, username, password, country/region, zip code, birthdate, and gender. You can create a new *outlook.com* or *hotmail.com* address. You will also need to provide a phone number or other method to be able to reset your password.
4. **Create** the account, accepting the Microsoft Services Agreement and privacy statement.

Task: Create an Azure DevOps Services Organization and Project

In this task your team will select someone to create a new Azure DevOps Services Organization and project.

Who will be performing this task? _____

1. Launch **Chrome**, navigate to https://dev.azure.com, and get started for free.

   You may be asked to review and accept the licensing terms. Don't create a new project yet.

   What is the name of your new organization? _____

2. With the new organization selected, select **Organization settings**.

3. Go to the **Overview** page.

   What is the organization *Name*? *URL*? _____

   If you are happy with this organization name and URL, then skip the next step.

4. Change the organization **Name** to a better, more meaningful name.

   Tip: Consider using a variation of your team's name.

   What is the new organization *URL*? _____

5. Share the **URL** with your team members.

6. Change the **Time zone** accordingly.

7. Go to the **Projects** page and create a **new project**:

   - Project name: **Fabrikam**
   - Visibility: **Private**
   - Advanced > Version Control: **Git**
   - Advanced > Work item Process: **Scrum**

Task: Configure the Team

In this task the person who created the project will configure the default team.

1. On the **Fabrikam** project page, select **Project settings**.

2. In **Project Settings**, go to the **Permissions** page.

3. With the **Fabrikam Team** selected, click the **Member of** link on the right.

4. Add the **[Fabrikam]\Project Administrators** group membership.

   Note: In practice, you should check with your administrator before adding all members of a team to the *Project Administrators* group.

5. Click the **Members** link.

   How many members are currently listed? _____

6. **Add** the **Microsoft Account** email addresses of your teammates.

Task: Join the Fabrikam Team

In this task the rest of the team members will access the newly created Azure DevOps project.

Note: You may receive an invitational email from Microsoft. You can ignore this.

1. Open your browser and navigate to the URL of your Azure DevOps project.

   Example: https://dev.azure.com/cicd2022/fabrikam

2. If prompted, enter your **Microsoft Account** and **password**.

   If you have any difficulties signing in, double-check your account and password. If that doesn't solve the problem, ask your colleague to double-check your team membership. If you are still blocked, consider going *Incognito* (Chrome) or check with the instructor.

Task: Update Your Profile

In this task everyone will update their individual profile.

1. In the upper-right corner, click User settings, and go to your profile.

   Note: Make sure not to click your avatar as that takes you to account manager options.

2. Verify that your full name and contact email address are accurate.

3. Change your image to something that better "represents" you.

   This is an optional step, but will improve your team's social experience.

4. Return to User settings, select **Notifications**, and unsubscribe from all notifications.

   This page shows subscriptions that you have created or have been created for you by an administrator or OOB (out-of-the-box). Active subscriptions are indicated with the State as *On*.

   Note: There may be a slight delay as you click each toggle, but in the end, your inbox will thank you.

EXERCISE 2 – IMPORT CODEBASE FROM GITHUB

Task: Import the Repository

In this task your team will select someone to import the codebase that will be used by Azure Pipelines.

Important: Only one person on your team needs to run this task.

Who will be performing this task? _____

1. Open your browser and navigate to the URL of your Azure DevOps **project**.

   Example: https://dev.azure.com/cicd2022/fabrikam

2. Go the **Files** page on the **Repos** hub.

3. From the **Fabrikam** repository dropdown at the top, select **Import repository**.

4. Enter https://github.com/accentient/fabrikamgames2022 for the Clone URL, enter **FabrikamGames** for the repository name, and click **Import**.

Task: Review the Repository or

In this task each team member, or pair of team members, will review the contents of the newly imported repository.

Dependencies

- The *FabrikamGames* repository exists and the codebase has been imported from GitHub

1. If necessary, navigate to the **Fabrikam** project.
2. Go to the **Files** page on the **Repos** hub.
3. From the repository dropdown at the top, make sure you are in the **FabrikamGames** repository.
4. Locate and click on the top-level **README.md** file.

   This markdown file explains the purpose of the repository. Feel free to use the *Preview* tab.
5. Review the folders and files under the **code** folder.

   This is a Visual Studio solution which contains three .NET Core projects: a TicTacToe game, a TicTacToe library, and accompanying unit tests.
6. From the ⋮ context menu to the right of the **code** folder, select ⤓ **Download as Zip**.

   Folders and files can be downloaded directly from Azure Repos. Folders are downloaded as zip files. Files can also be uploaded directly to the repository as well.

EXERCISE 3 – CREATE AND USE A BUILD PIPELINE or

Task: Use the Visual Designer to Create a Build Pipeline

In this task each team member, or pair of team members, will create a build pipeline to automatically build the .NET Core TicTacToe projects.

1. Navigate to the **Pipelines** page in the **Pipelines** hub.
2. **Create** a new pipeline**.**
3. Select to **Use the classic editor** (visual designer).
4. Ensure **Azure Repos Git** is selected as the source and **FabrikamGames** as the repository.
5. Instead of selecting a template, select an **Empty job**.
6. Review the **Pipeline** settings.

   The hosted Azure Pipelines agent pool is associated with *your Azure DevOps Services* account. As the name suggests, it is hosted in Azure and managed automatically by Microsoft.

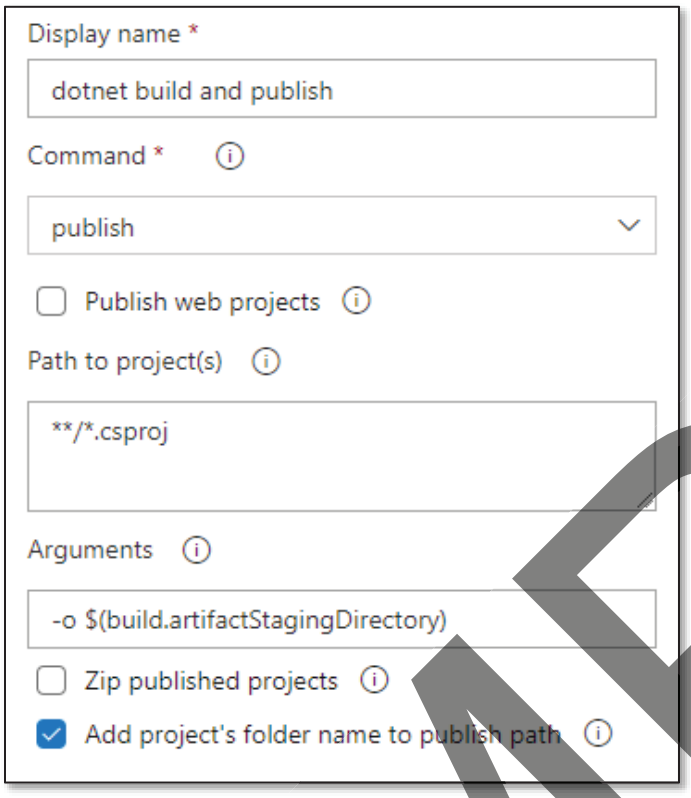
   What *Agent Specification* is selected? _____
7. Change the name to ***XX*-Fabrikam.Games.NoTests** (where *XX* = your initials).
8. Ensure the Agent specification is set to **windows-2022**.

   Note: This may or may not be the same as *windows-latest*.
9. In the **Agent job 1** section on the left, click +, and add a **.NET Core** task.
10. Select the new .NET Core task and set the following properties:

    - Display name: **`dotnet build and publish`**
    - Command: **`publish`**
    - Publish web projects: <cleared>
    - Path to project(s): **`**/*.csproj`**
    - Arguments: **`-o $(build.artifactStagingDirectory)`**
    - Zip published projects: <cleared>

    The dotnet publish command will implicitly build the project(s) before publishing, unless a *--no-build* option is specified. For more readability, we could have added two .NET Core tasks – one to build and one to publish.

Your settings should look something like this:



11. Add a **Publish build artifacts** task and position it below the .NET Core step.

    This task takes the published output from the .NET Core task and makes it available in the pipeline. You do not need to change any of the default settings.

12. On the **Options** page, enter a Description of **Builds the TicTacToe projects**.

    Will work items be automatically linked to this build? _____

    Will a work item be automatically created if the build fails? _____

13. From the **Save & queue** dropdown, select **Save**.

14. Enter a meaningful comment and click **Save**.

Task: Run the pipeline

In this task each team member, or pair of team members, will run the pipeline to build the TicTacToe solution.

1. Return to the **Pipelines** page in the **Pipelines** hub.

   Do you see *your* new build pipeline? _____

   Note: If you don't see it in the *Recent* list, you may have to click *All* at the top.

2. Identify and select another team member's build pipeline to use.

   Whose build pipeline will you use? _____

   Note: If you are by yourself or your teammate(s) have not caught up yet, select your own pipeline.

3. **Run** the pipeline using default settings.

   Notice that you can override the buildconfiguration variable's default value.

4. Click the **Agent job 1** at the bottom to view progress.

   What is your *build number* (e.g. *20220201.1*)? _____

   Did the build succeed? How long did it take? _____

   Note: If you see an error message about "No hosted parallelism" this means that your Azure DevOps organization has not been granted free parallelism. Someone will need to complete this form: https://aka.ms/azpipelines-parallelism-request. Unfortunately, since it takes 2-3 business days to complete the upgrade, class might be over. Check with your instructor on what to do in this case. For more, read this blog post: https://bit.ly/AzurePipelinesGrant.

5. Click the **Initialize job** step.

   This shows details from this specific step.

6. Click the **dotnet build and publish** step.

7. Click **View raw log** and review the detailed steps.

   This will open another tab showing much more detail. You can see the specific settings and commands for each task. These pages are very useful when you are troubleshooting.

   Note: You can ignore any compiler warnings. Also, if you want to suppress all compiler warnings, you can set the MSBuild Arguments to */p:WarningLevel=0* on the Visual Studio build task.

8. Return to the original tab and click ← to go back to the pipeline summary.

   What time did the pipeline start? What was the total duration? _____

9. In the Related section, click **1 published; 1 consumed**.

   These are the folders and files that were built and generated as part of the build process.

10. Expand all of the folders and review the types of files in each one.

11. To the right of the **drop** folder, click the ⋮ more options menu, and select **Download artifacts**.

12. Click ← to return to the summary.

13. From the ⋮ menu in the upper right, select **Download logs**.

    This will download the log files in a zip file. Feel free to explore them.

Task: Clone a Build Pipeline

In this task each team member, or pair of team members, will clone a pipeline, make a change, queue a new build, and inspect the results.

1. Return to the **Pipelines** page in the **Pipelines** hub.

2. Find and select your ***XX*-Fabrikam.Games.NoTests** pipeline.

   Note: If you don't see it in the *Recent* list, you may have to view *All* pipelines.

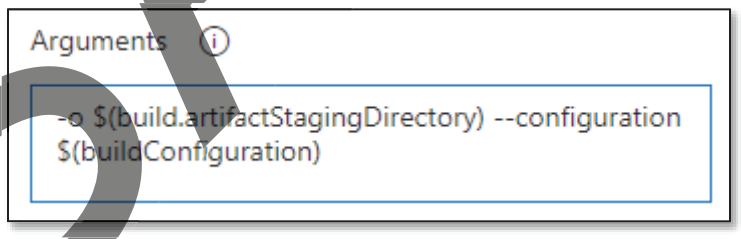
3. From the ⋮ menu in the upper right, select **Clone**.

4. Change the name to ***XX*-Fabrikam.Games.Configuration.NoTests** (where *XX* = your initials).

5. Select the .NET Core task and add another command to the Arguments field:

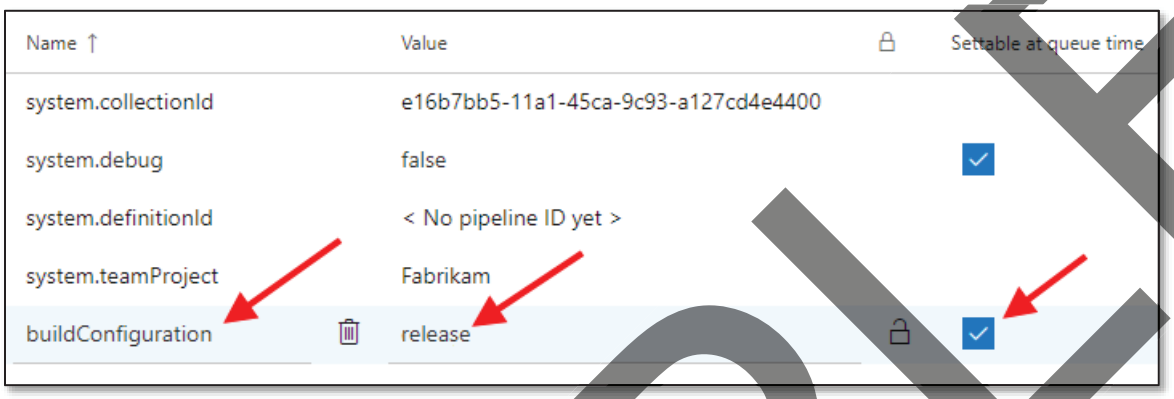
   `--configuration $(buildConfiguration)`

   The Arguments should look like this:



6. Go to the **Variables** page.

7. Add a new pipeline variable named **buildConfiguration** with a value of **release** and make it settable at queue time, like this:



8. From the **Save & queue** dropdown, select **Save**.

9. Enter a meaningful comment and click **Save**.

10. Return to the **Pipelines** page in the **Pipelines** hub.

11. Select and **Run** your new "configuration" pipeline using default values.

    How long did it take this build to complete? Was it faster than the previous one? _____

    Tip: You can always click the *Agent Job* at the bottom to view progress.

12. Run the "configuration" pipeline again, but this time click **Variables** and change the buildConfiguration value to **debug**.

    Note: You may not be able to see the subtle differences between a debug and a release build.

EXERCISE 4 – INCLUDE AUTOMATED TESTS IN A PIPELINE or

Task: Run Tests in Your Pipeline

In this task each team member, or pair of team members, will clone a pipeline, add automated tests, run a new build, and inspect the results.

1. Return to the **Pipelines** page in the **Pipelines** hub.

2. Find and select your original ***XX*-Fabrikam.Games.NoTests** build pipeline.

   Note: If you don't see it in the *Recent* list, you may have to view *All* pipelines.

3. From the ⋮ menu in the upper right, select **Clone**.

4. Change the name to ***XX*-Fabrikam.Games.Tests** (where *XX* = your initials).

5. Select the **.NET Core** task and set the following properties:

   - Display name: **`dotnet build`**
   - Command: **`build`**
   - Arguments: <cleared>

   For this pipeline, we will break down our build, test, and publish into three separate steps.

6. Add another **.NET Core** task, position it below the *dotnet build* .NET Core step, and set the following properties:

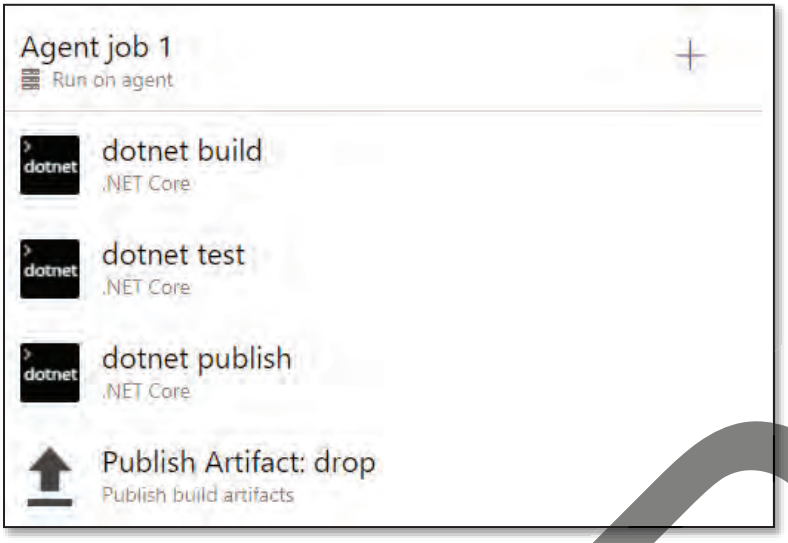
   - Command: **`test`**
   - Path to project(s): **`**/TicTacToeLibTests.csproj`**
   - Arguments: **`--no-build`**

   The dotnet test command is used to execute unit tests. It will also implicitly build the project(s), unless a *--no-build* option is specified, like we are doing in this step.

7. Add another **.NET Core** task, position it below the *dotnet test* .NET Core step, and set the following properties:

   - Command: **`publish`**
   - Publish web projects: <cleared>
   - Path to project(s): **`**/*.csproj`**
   - Arguments: **`--no-build -o $(build.artifactStagingDirectory)`**
   - Zip published projects: <cleared>

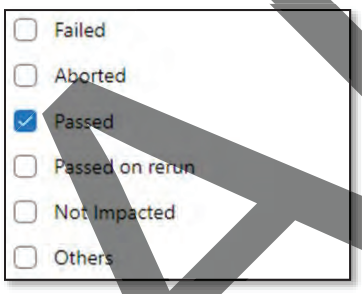
Your pipeline should now look like this:



8. From the **Save & queue** dropdown, select **Save & queue**.
9. Enter a meaningful comment and click **Save and run**.
10. When the build completes, go to the **Summary** page and view the **Tests**.

    How many total tests were *run*? How many *passed*? _____

11. In the bottom section change the outcome filter to show only **Passed** tests.



You should see each of the passing unit tests at this point. You can click on a test to see more information about it which is very useful when you are troubleshooting.

Task: Calculate Code Coverage in Your Pipeline

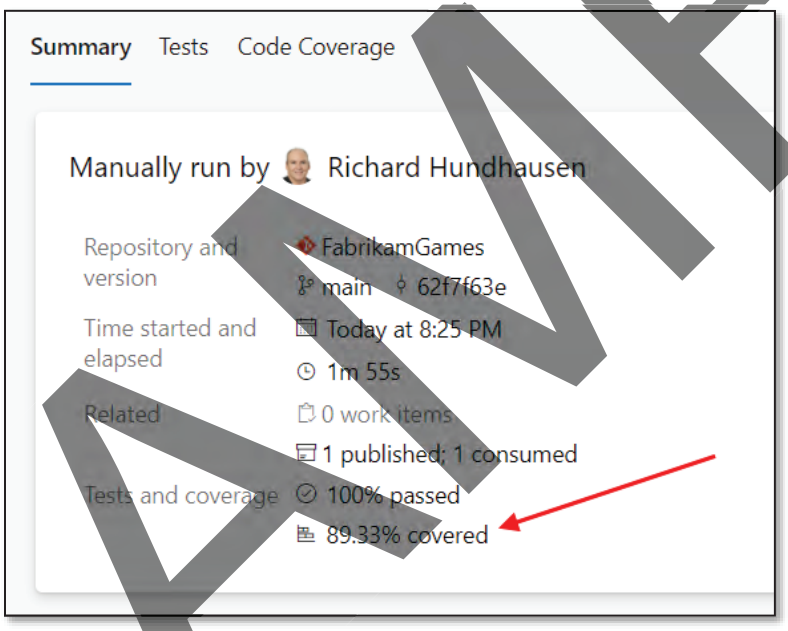
In this task each team member, or pair of team members, will clone a pipeline, configure it to calculate code coverage in the form of a Visual Studio .coverage file, run a new build, and inspect the results.

1. Return to the **Pipelines** page in the **Pipelines** hub.
2. Find and select your ***XX*-Fabrikam.Games.Tests** pipeline.
3. From the ⋮ menu in the upper right, select **Clone**.
4. Change the name to ***XX*-Fabrikam.Games.Tests.Coverage** (where *XX* = your initials).
5. Select the **dotnet test** .NET Core task and add the following to the Arguments field:

   ```
   --collect "Code Coverage"
   ```

6. **Save & queue** the pipeline, specifying a meaningful comment.

   When the build completes, what coverage percentage is displayed on the summary? _____



7. Go to the **Code Coverage** tab and click **Download code coverage results**.

   If you have Visual Studio Enterprise edition, you can open the .coverage file for analysis in the Code Coverage Results window. There are other, open-source solutions for computing and reporting code coverage as you will see in the next task.

Task: Calculate Code Coverage Using Coverlet in Your Pipeline (Optional)

In this task each team member, or pair of team members, will clone a pipeline, configure it to calculate code coverage using Coverlet, run a new build, and inspect the results. For more information on Coverlet, visit https://github.com/coverlet-coverage/coverlet.

1. Return to the **Pipelines** page in the **Pipelines** hub.

2. Find and select your ***XX*-Fabrikam.Games.Tests.Coverage** pipeline.

3. From the ⋮ menu in the upper right, select **Clone**.

4. Change the name to ***XX*-Fabrikam.Games.Tests.Coverlet** (where *XX* = your initials).

5. Select the **dotnet test** .NET Core task and replace the existing Arguments with this:

   ```
   --collect:"XPlat Code Coverage"
   ```

6. Add a **Publish code coverage results** task, position it below the *dotnet test* .NET Core step, and set the following properties:

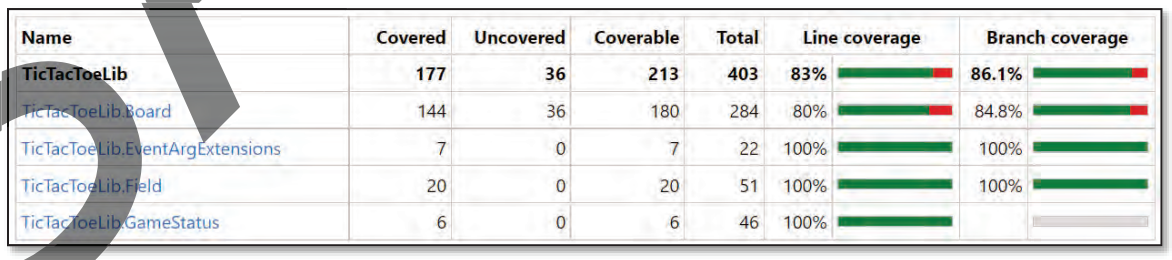
   - Display name: **Publish code coverage report**
   - Code coverage tool: **Cobertura**
   - Summary file: **`$(Agent.TempDirectory)/*/coverage.cobertura.xml`**

   This task publishes code coverage results produced when running tests to Azure Pipelines in order to obtain coverage reporting. The task supports popular coverage result formats such as Cobertura and JaCoCo.

7. **Save & queue** the pipeline, specifying a meaningful comment.

8. When the build completes, go to the **Code Coverage** tab, view the Summary, and drill down into some of the TicTacToeLib areas.

   Instead of just being able to download a .coverage file (which requires Visual Studio Enterprise to open), you now have an HTML report showing code coverage details. Reports are automatically generated and are rendered with appropriate using the open source tool *ReportGenerator*.



Name	Covered	Uncovered	Coverable	Total	Line coverage	Branch coverage
TicTacToeLib	**177**	**36**	**213**	**403**	**83%**	**86.1%**
TicTacToeLib.Board	144	36	180	284	80%	84.8%
TicTacToeLib.EventArgExtensions	7	0	7	22	100%	100%
TicTacToeLib.Field	20	0	20	51	100%	100%
TicTacToeLib.GameStatus	6	0	6	46	100%	

Task: Explore Pipeline Analytics

In this task each team member, or pair of team members, will explore the analytics available on a build pipeline.

1. Return to the **Pipelines** page in the **Pipelines** hub.
2. Find and select your **Coverage** build pipeline.
3. Go to the **Branches** tab.

   In the *History* section, how many green bars are showing? _____

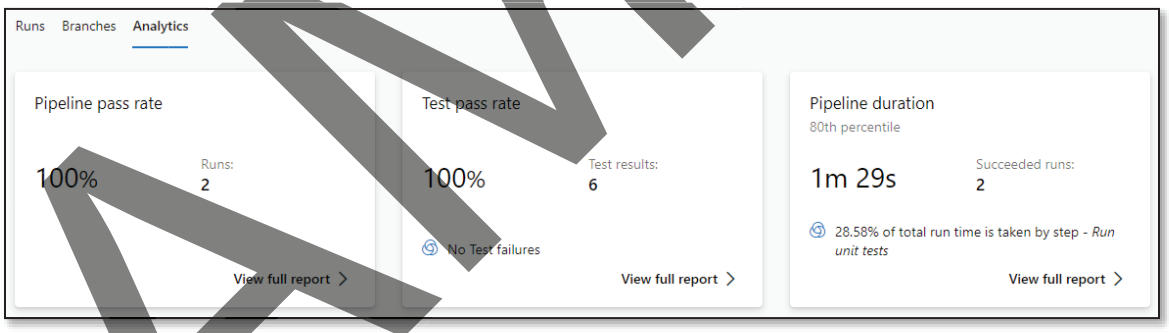
4. Hover over each green bar.

   What was the duration of the first run? The second run? _____

   This makes it easy to see that the baseline build took longer than the second "impacted" build.

5. Go to the **Analytics** tab.

   You should see three analytics here: Pipeline pass rate, Test pass rate, and Pipeline duration. The analytics show the build pass rate and the failure trend, the tasks failure trend to provide insights on which task in the pipeline is contributing to the maximum number of failures, and the trend of time taken for a pipeline to run as well as which tasks in the pipeline are taking the most amount of time. Here is an example:



6. Select the **Pipeline duration** analytic.

   Which task/step had the longest duration? What was the percentage? _____

   You can learn more about the various pipeline analytics here: https://bit.ly/3KXJcFs. Feel free to review analytics on other pipelines.

EXERCISE 5 – PRACTICE CONTINUOUS INTEGRATION or

Task: Enable Continuous Integration

In this task each team member, or pair of team members, will clone a build pipeline and configure it to trigger on a push to the repository.

1. Return to the **Pipelines** page in the **Pipelines** hub.
2. Find and select your ***XX*-Fabrikam.Games.Tests** pipeline.
3. From the ⋮ menu in the upper right, select **Clone**.
4. Change the name to ***XX*-Fabrikam.Games.CI** (where *XX* = your initials).
5. Go to the **Triggers** tag.
6. Select **Enable continuous integration**.

   Notice that you can enable *Batch changes*. This is helpful when you have many active team members and you want to reduce the number of running builds. With *Batch changes*, when a build is running, Azure Pipelines waits until the build is completed and then queues another build of all the pending commits.

7. Add a **Path Filter** to **Include** the **code** path.

   You can set *Branch* and *Path* filters to keep commits in other branches and folders from triggering a build. These filters are case sensitive. If you *don't* set a path filter, then the root folder of the repo is implicitly included by default. For more information on filters, visit https://bit.ly/32JjOii.

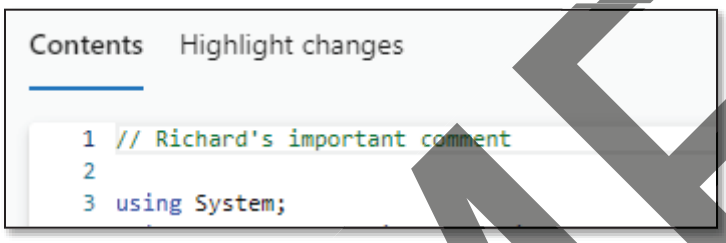
8. From the **Save & queue** dropdown, select **Save.**
9. Enter a meaningful comment and click **Save**.

Task: Practice Continuous Integration

In this task your team will self-organize and practice Continuous Integration.

Note: Try to only have one team member at a time editing the file. Also, keep in mind that a commit from a team member can trigger *multiple* builds.

1. Go to the **Files** page of the **Repos** hub.
2. Navigate to the **code** > **TicTacToe** subfolder
3. Click the **Program.cs** file.
4. Click **Edit**.
5. Add a personal comment at the top of the file, like this example:



6. Click **Commit**.
7. Enter a meaningful commit comment and click **Commit**.
8. Return to the **Pipelines** page in the **Pipelines** hub.

   You should see your CI build running. Click it to monitor it and view the results. Depending on how many team members also created pipelines and committed changes, there may be additional pipeline runs queued up.

   If necessary, have 1 or 2 more team members commit a change to the file to practice Continuous Integration. You can also try making changes to the README.md file in the root to see how the trigger filter will ignore it.

9. **Pause** or **Delete** any duplicate CI pipelines.

   After practicing Continuous Integration, your team should self-organize and *Pause* or *Delete* any duplicate pipelines to cut down on the noise and increase performance.

EXERCISE 6 – USE YAML TO CONFIGURE A PIPELINE (OPTIONAL)

Task: Create a YAML-Based Build Pipeline

The YAML acronym used to stand for Yet Another Markup Language, but the maintainers renamed it to *YAML Ain't Markup Language* to place more emphasis on its data-oriented features.

In this task each team member, or pair of team members, will create a YAML-based pipeline to build the TicTacToe solution.

1. Go to the **Pipelines** page in the **Pipelines** hub.
2. Create a **New pipeline**.
3. Select **Azure Repos Git *YAML***.
4. Select the **FabrikamGames** repository.
5. Select the **Starter pipeline** template and review the default YAML file contents.

   What vmImage is being used by default? _____
6. Click azure-pipelines.yml and rename to ***XX*-fabrikamgames.yml** (where *XX* = your initials), like this example:

7. Make the following changes to the YAML file:
   - trigger: **none** (and remove the **- main** line below)

```
# Starter pipeline
# Start with a minimal pipeline that you can customize to build and deploy your code.
# Add steps that build, run tests, deploy, and more:
# https://aka.ms/yaml

trigger: none

pool:
  vmImage: ubuntu-latest
```

   For more information on YAML schema specifics, visit https://aka.ms/yaml.
8. From the **Save and run** dropdown, select **Save**.

9. Change the commit message to **My first YAML build pipeline** and click **Save**.

   These changes will be committed directly to the master branch.

10. Return to the **Pipelines** page in the **Pipelines** hub and view **All** pipelines.

    Do you see the new *FabrikamGames* pipeline? _____

    Unfortunately, it's not apparent which pipelines are YAML-based ones. You could adopt a naming convention or use folders.

11. From the ⋮ menu to the right of the **FabrikamGames** pipeline, select **Rename/move**.

12. Change the name to ***XX*-FabrikamGames.YAML** (where *xx* = your initials) and click **Save**.

13. Go to the **Files** page of the **Repos** hub.

    Do you see your new *XX*-fabrikamgames.yml file? _____

14. Click your ***XX*-fabrikamgames.yml** file.

    As you can see, the pipeline configuration is just code. It can be changed and version-controlled like any other file in the codebase. Don't make any changes at this point.

15. Return to the **Pipelines** page in the **Pipelines** hub and view **All** pipelines.

16. **Run** your new ***XX*-FabrikamGames.YAML** pipeline using default values.

17. When the build has completed, review the "Run" steps in the job.

    Do you see the "Hello, world!" output? _____

18. Return to the **Pipelines** page and **Edit** your ***XX*-FabrikamGames.YAML** pipeline.

    This will, in turn, edit the underlying *XX-fabrikamgames.yml* file. This time, rather than having to memorize the YAML statements you want to type, you can use an assistant to help write the script.

19. Change the vmImage value to **windows-2022** and remove both *script* steps, but leave the "steps:"

```
pool:
  vmImage: windows-2022

steps:
- script: echo Hello, world!
  displayName: 'Run a one-line script'

- script: |
    echo Add other tasks to build, test, and deploy your project.
    echo See https://aka.ms/yaml
  displayName: 'Run a multi-line script'
```

20. Place the cursor at the bottom of the file, on a blank line just below *steps:*

21. Review the list of **Tasks** on the right.

    You may have to click *Show assistant* to see these tasks.

22. Select the **.NET Core** task, specify the following, and then **Add** it to the script.

    - Path to project(s): **`**/*.csproj`**

    This will add the related snippet to the YAML file.

23. Manually add the following after the - *task: DotNetCoreCLI@2* line:

    `displayName: 'dotnet build'`

    Important: Whitespace is an important part of YAML's formatting. YAML documents are structured using indentation – with spaces, not tabs. In other words, you must line up "displayName" with the "t" in task. You script should look like this now:

```
steps:
Settings
- task: DotNetCoreCLI@2
  displayName: 'dotnet build'
  inputs:
    command: 'build'
    projects: '**/*.csproj'
```

24. **Save** and commit your changes.

25. Return to the **Pipelines** page in the **Pipelines** hub and view **All** pipelines.

26. Run your updated YAML pipeline using the default values.

    The build should complete successfully.

27. Return to the **Pipelines** page and **Edit** your ***XX*-FabrikamGames.YAML** pipeline.

28. Add a blank line at the bottom of the YAML.

29. Open a second browser tab and **Edit** your ***XX*-Fabrikam.Games.Tests** pipeline in the new tab.

30. In the second browser tab, select the **dotnet test .NET Core** task and click **View YAML**.

31. Copy the YAML – minus the "steps:" directive – to the clipboard, return to the first browser tab and paste the content at the bottom of the YAML.

32. Repeat the above steps to copy and paste the remaining YAML snippets.

    Your YAML steps should look something like this:

```
steps:
Settings
- task: DotNetCoreCLI@2
  displayName: 'dotnet build'
  inputs:
    command: 'build'
    projects: '**/*.csproj'
Settings
- task: DotNetCoreCLI@2
  displayName: 'dotnet test'
  inputs:
    command: test
    projects: '**/TicTacToeLibTests.csproj'
    arguments: '--no-build'
    testRunTitle: 'Cool unit tests'
Settings
- task: DotNetCoreCLI@2
  displayName: 'dotnet publish'
  inputs:
    command: publish
    publishWebProjects: false
    projects: '**/*.csproj'
    arguments: '--no-build -o $(build.artifactStagingDirectory)'
    zipAfterPublish: false
- task: PublishBuildArtifacts@1
  displayName: 'Publish Artifact: drop'
```

33. **Save**, **Commit**, and **Run** your YAML pipeline.