Book Online Order System

- Pipeline
- Automated Testing
- CI/CD
- Three concepts in the project

"The core of the '41026 Advanced Software Development' course at UTS aims to teach students three key knowledge points. The first is the use of pipelines, understanding pipelines, and what role pipelines can play in software development. The second is automated testing, which involves conducting certain tests on software during the project development process. The third is CI/CD, which stands for Continuous Integration and Continuous Deployment. Moving forward, I will provide necessary insights on these three points (from my understanding)."

Pipeline（Concepts）

"In the process of software development, a 'Pipeline' can be understood as a code conveyance channel. Its primary function is to automate the transfer and processing of code, supporting the continuity and efficiency of software development.

During the lifecycle of a software project, the development process typically goes through multiple release stages or iterations. Each stage may involve the implementation of new requirements and features for the software. For example, the first phase might complete login and registration functions, while the second phase might introduce a shopping cart feature. As each stage is completed, the code needs to be updated and maintained accordingly.

In traditional software development models, these code updates are usually done manually. The disadvantage of this approach is that it increases the workload, and developers also need to track and check which blocks of code have been modified, which is both time-consuming and prone to errors.

The introduction of Pipelines greatly improves this process. It automates the management of code transmission and deployment, ensuring that new code from each iteration is integrated and deployed efficiently and accurately to the production environment. This not only reduces the burden on developers but also increases the speed and reliability of the entire development process. In short, Pipelines play a crucial 'conveyor belt' role in software development, ensuring the continuous integration and delivery of code."

Automating Testing（Concepts）

"Automated testing, as the name implies, is a method of testing that is executed automatically. Its main purpose is to reduce the workload of manual testing. In traditional testing methods, testers need to individually check each function completed by the developers, such as registration and login features. However, in automated testing, this process is simplified.

In the automated testing workflow, developers first need to write test code and configure the corresponding YAML files in the pipeline. When the pipeline is triggered, it automatically runs the code specified in the YAML files. This code, in turn, executes test scripts to check the newly developed features. Through this method, the testing process is no longer dependent on manual operations but is entirely controlled by code, achieving automation of testing. This not only improves the efficiency of testing but also ensures the consistency and accuracy of the tests."

Examples:

1. **Setting up and Starting**: Just like selecting a washing program and starting the washing machine, automated testing needs to be correctly configured and initiated at the beginning.
2. **Automated Task Execution**: The washing machine automatically completes washing, rinsing, and spinning, similar to how automated testing scripts execute predefined test cases.
3. **Consistency and Reliability**: The washing machine performs the washing process in the same way every time, ensuring consistency in the washing results. Similarly, automated testing ensures that after each code change, tests are executed in the same manner, providing reliable test results.
4. **Efficiency and Time-Saving**: Using a washing machine is more efficient than hand washing, saving time and effort. Automated testing similarly increases the efficiency of the testing process, reducing the need for manual testing.

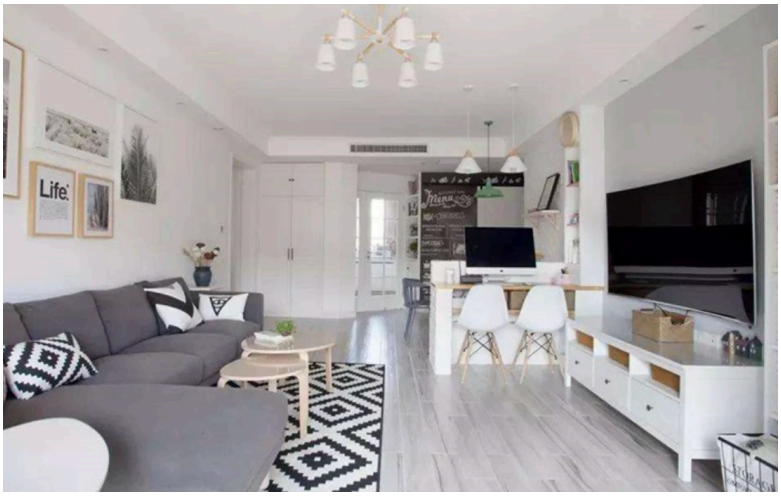
CI/CD（Concepts）

"The full names of CI/CD are Continuous Integration and Continuous Delivery. These two concepts are key practices in software development, aimed at improving the efficiency and quality of software development and release.

First, let's delve into Continuous Integration. During the software development process, it is usually divided into multiple iteration stages. In each stage, developers write code according to functional requirements. Once the functional requirements of a stage are developed, these newly developed features need to be integrated into the existing code. This integration process is not just a simple merging of code; it also includes a series of automated tests to ensure that the addition of new code does not disrupt the functionality of the existing system. Therefore, Continuous Integration can be understood as an ongoing process where newly developed code is frequently and automatically merged with the main codebase, and thoroughly tested before the merge."

Examples: Imagine you and your friends are preparing a big meal together. Each person is responsible for a different dish. In the Continuous Integration approach, as soon as someone finishes their part of the cooking, they bring it to the communal table. This way, if a dish doesn't taste good or there's a problem with how it's cooked, everyone can immediately notice and make adjustments. This process is like Continuous Integration in software development, where developers frequently merge code changes into the main branch and immediately test after merging to ensure these changes do not disrupt the existing system.

Next, let's explore Continuous Delivery. Similar to Continuous Integration, Continuous Delivery focuses on how to efficiently and reliably deliver new versions of software to users or stakeholders. In traditional software development models, releasing software is often a big event, possibly involving months or even years of development work. However, under the Continuous Delivery model, software releases become more frequent and automated. For example, software might be divided into multiple versions, such as release 0, release 1, release 2, etc. As soon as a version passes all tests and completes integration, it can be automatically and rapidly deployed to the production environment or delivered to users. This approach ensures rapid iteration and continuous improvement of the software.



Examples: Imagine you are doing a home renovation, but instead of waiting to reveal the entire house once it's completely done, you choose a phased approach. First, you complete and showcase the living room renovation. Once the living room is done, you start using it while beginning the bedroom renovation. The completion of each room is a mini "release," allowing you to immediately enjoy and use the updated space while continuing work on other parts. This is like Continuous Delivery in software development, where new

features and improvements are gradually delivered to users soon after development, rather than waiting until all features are fully developed before releasing.

In summary, CI/CD is about implementing automation and continuous improvement in the software development process. Through Continuous Integration, we ensure the quality and consistency of code; and through Continuous Delivery, we ensure that software can reach users quickly and frequently, thereby accelerating the feedback loop and product iteration. Continuous Integration emphasizes frequently and incrementally merging and testing changes, while Continuous Delivery focuses on gradually and continuously delivering these changes to the end-users.

Three Concepts In The Project

Automated Testing

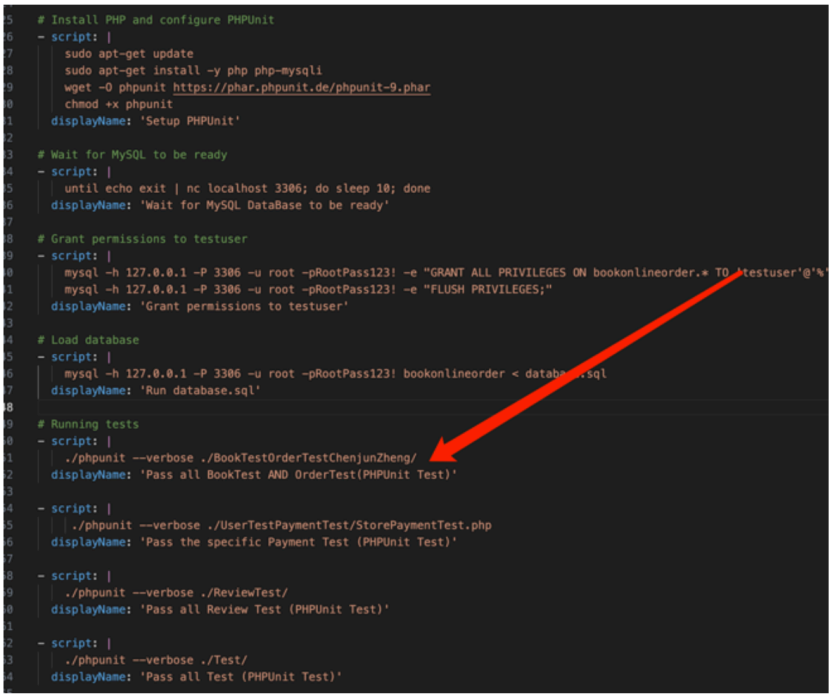
"In this project, we utilized the Pipeline service provided by Azure, achieving tight integration with the GitHub repository. The core of this integration is that whenever the GitHub repository is updated, Azure Pipeline automatically triggers and executes a series of predefined tasks, which are configured through a YAML file.

The project adopts a standard technology stack for the frontend (HTML, CSS, JavaScript) and PHP and MySQL for the backend technology. For the backend code, we chose PHPUnit as the testing framework to ensure the quality and stability of the code."

```php
<?php
//Importing PHPUnit's TestCase class for creating test cases
use PHPUnit\Framework\TestCase;

//import addToCart.php for test
require_once 'addToCart.php';

//Define the AddToCartTest class to test the addToCart function, inherited from PHPUnit's TestCase class
class AddToCartTest extends TestCase
{
    public function testAddToCart()
    {
        $bookId = 1;
        $existingCartId = null;

        // Call the addToCart function and pass in the book ID, the existing shopping cart ID and a Boolean value
         // Store the returned result in the $output variable
        $output = addToCart($bookId, $existingCartId, false);

        $this->assertEquals("Book added to cart successfully", $output);
    }
}
```

"After the test scripts are written, the next step is to write the YAML configuration file. The purpose of this file is to guide Azure Pipeline to execute specific steps, such as installing dependencies, running PHPUnit test scripts, etc. Through this method, we can precisely control every execution stage of the Pipeline."

Test Code Run Successfully Pipeline

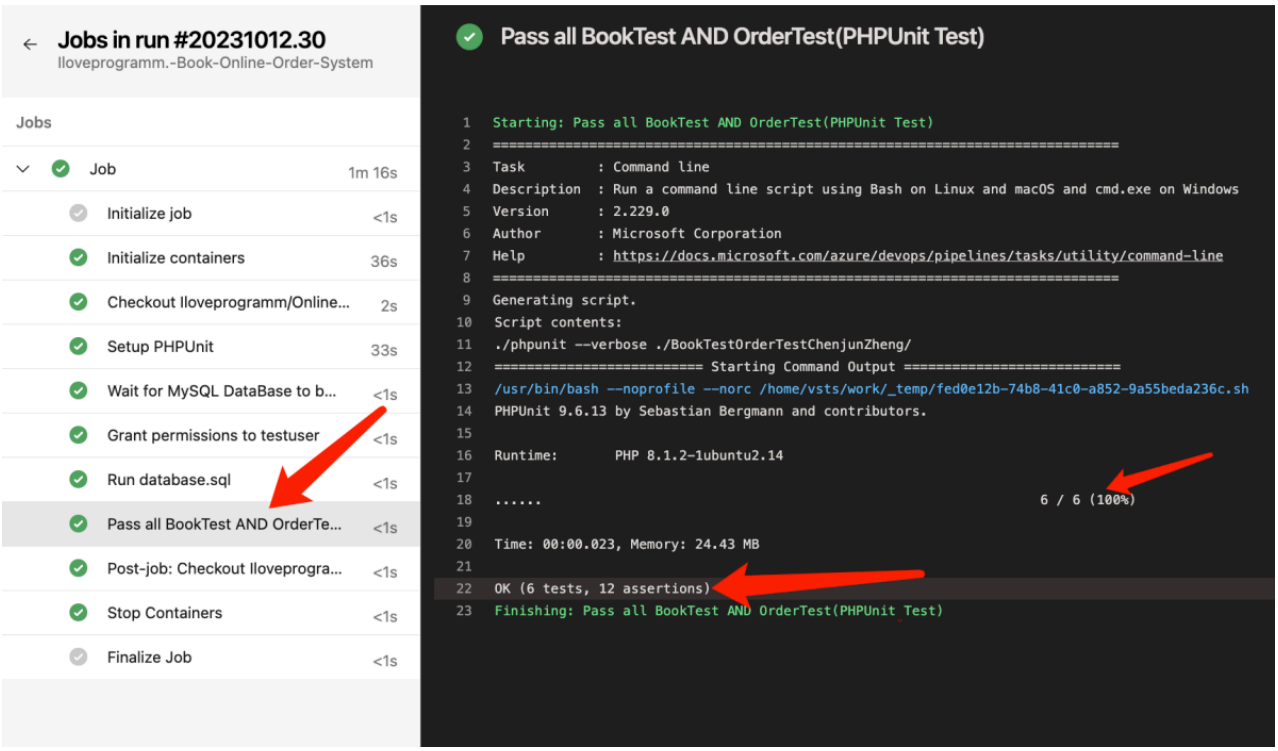


```
# Install PHP and configure PHPUnit
- script: |
    sudo apt-get update
    sudo apt-get install -y php php-mysqli
    wget -O phpunit https://phar.phpunit.de/phpunit-9.phar
    chmod +x phpunit
  displayName: 'Setup PHPUnit'

# Wait for MySQL to be ready
- script: |
    until echo exit | nc localhost 3306; do sleep 10; done
  displayName: 'Wait for MySQL DataBase to be ready'

# Grant permissions to testuser
- script: |
    mysql -h 127.0.0.1 -P 3306 -u root -pRootPass123! -e "GRANT ALL PRIVILEGES ON bookonlineorder.* TO 'testuser'@'%'
    mysql -h 127.0.0.1 -P 3306 -u root -pRootPass123! -e "FLUSH PRIVILEGES;"
  displayName: 'Grant permissions to testuser'

# Load database
- script: |
    mysql -h 127.0.0.1 -P 3306 -u root -pRootPass123! bookonlineorder < database.sql
  displayName: 'Run database.sql'

# Running tests
- script: |
    ./phpunit --verbose ./BookTestOrderTestChenjunZheng/
  displayName: 'Pass all BookTest AND OrderTest(PHPUnit Test)'

- script: |
    ./phpunit --verbose ./UserTestPaymentTest/StorePaymentTest.php
  displayName: 'Pass the specific Payment Test (PHPUnit Test)'

- script: |
    ./phpunit --verbose ./ReviewTest/
  displayName: 'Pass all Review Test (PHPUnit Test)'

- script: |
    ./phpunit --verbose ./Test/
  displayName: 'Pass all Test (PHPUnit Test)'
```

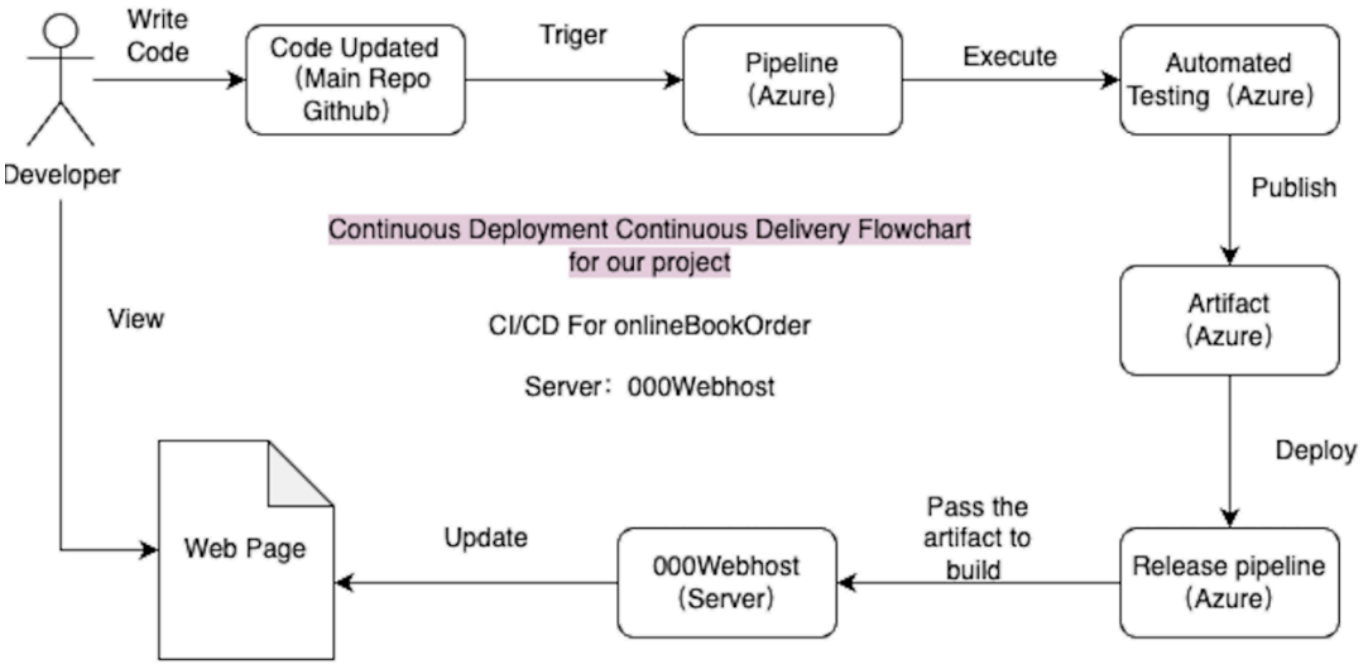
"Each time the code in the GitHub repository is updated, Azure Pipeline automatically executes the related testing and deployment tasks according to the instructions in the YAML file. This method of automated testing greatly improves development efficiency and reduces the need for manual intervention. Developers no longer need to manually run each test but rely on the Pipeline to automatically complete these tasks.

The advantage of this process is that it not only automates the testing process but also ensures that every code update undergoes rigorous quality checks. In this way, throughout the development process, each feature is repeatedly verified to ensure it works as expected, thereby guaranteeing the quality and stability of the final product.

In summary, by integrating Azure Pipelines with GitHub and using YAML files to precisely control the testing and deployment process, we have created an efficient and reliable automated testing environment, which is crucial for modern software development."



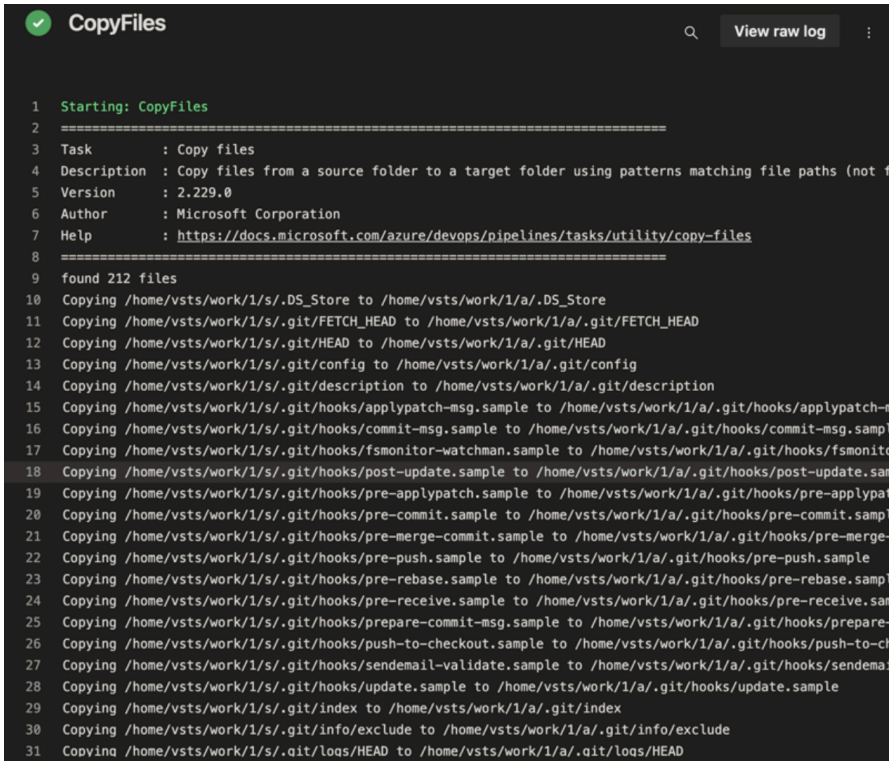
CI/CD

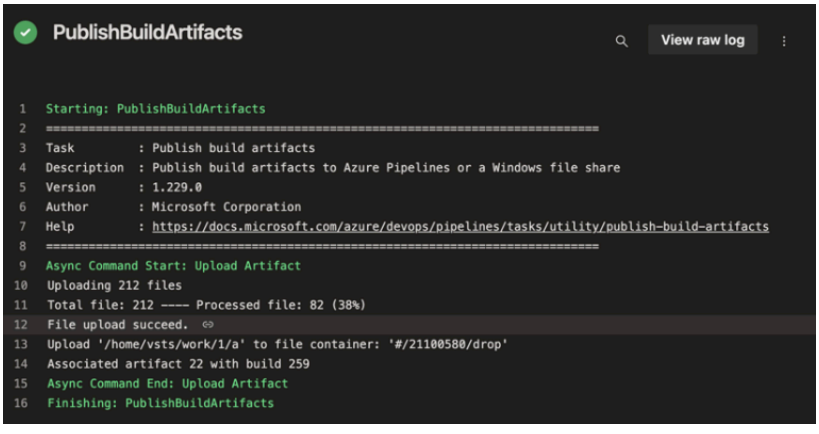


In order to achieve Continuous Delivery (CD) and Continuous Integration (CI), we will be using the Azure Pipelines service. This service is connected to the main branch on GitHub, and any changes to the main branch will trigger the Pipeline, thereby creating a new workflow. The first step in this process is to generate an Artifact. An Artifact can be thought of as a package that contains all the files from the code repository. Comparing it to the operation of a restaurant, when chefs have prepared the dishes, they need to present them at the front, which is akin to releasing an Artifact. Subsequently, the waitstaff delivers these dishes to the customers, which corresponds to deploying the Artifact to the server. Therefore, we need to create a release Pipeline, in which we will package and save all code files into an Artifact, and then deploy the Artifact to the target environment through the Pipeline.

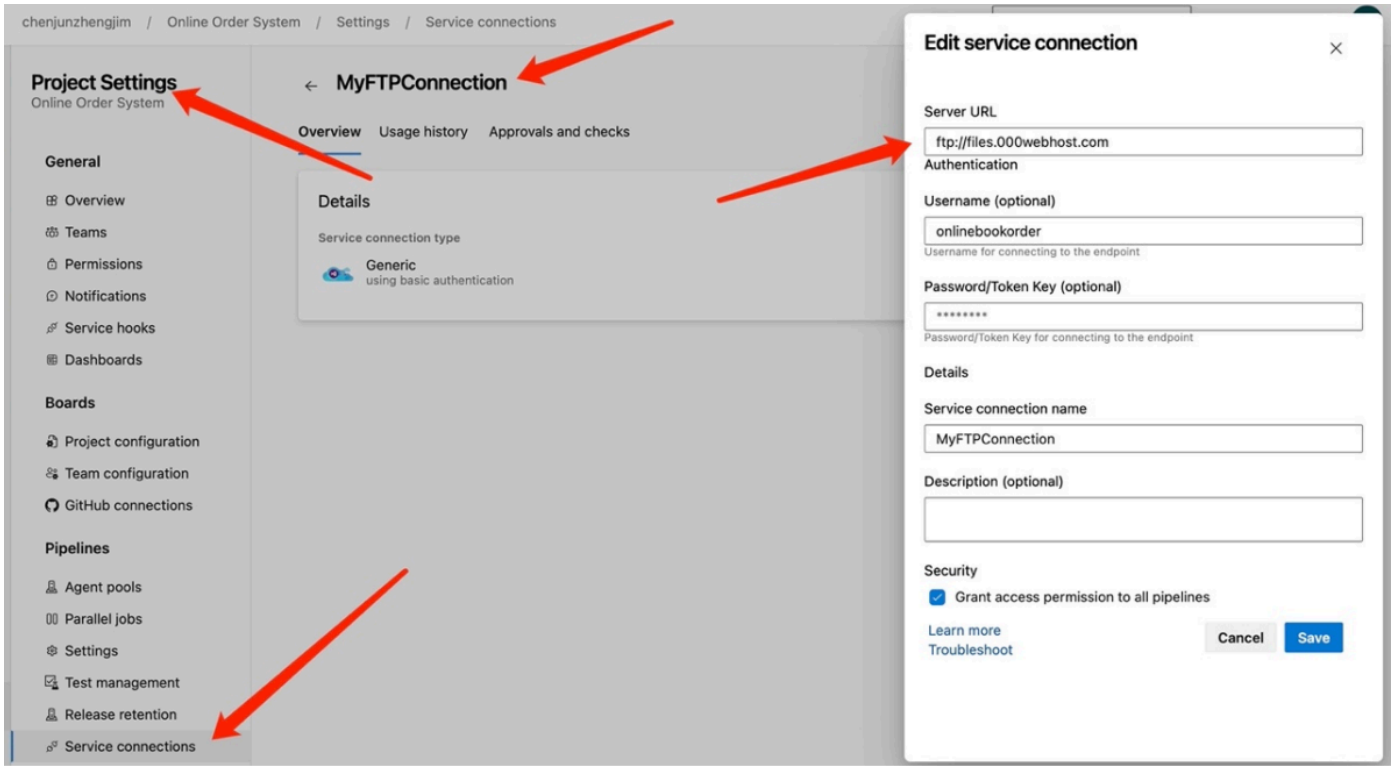
```yaml
- task: CopyFiles@2
  inputs:
    SourceFolder: '$(Build.SourcesDirectory)'
    Contents: '**'
    TargetFolder: '$(Build.ArtifactStagingDirectory)'

- task: PublishBuildArtifacts@1
  inputs:
    PathtoPublish: '$(Build.ArtifactStagingDirectory)'
    ArtifactName: 'drop'
    publishLocation: 'Container'
```

These two tasks serve to save and store the code files from the repository, and then to release the Artifact





After the release of the Artifact, the next step is to set up a connection with the target server to ensure the Pipeline knows where to deploy the Artifact. To do this, within the Azure project's "Project Settings," we need to locate the "Service Connections" section and configure the server's URL there. By doing this, the service can be referenced in the Pipeline's configuration file without directly exposing any sensitive URL information in the yaml file. This approach is taken to enhance overall security and ensure that critical connection details are properly managed.



Once the server URL configuration is completed, we are nearing the end of the workflow. This stage involves editing the Pipeline's yaml configuration file to release the Artifact onto the server we previously set up. We need to accurately fill in the server's login credentials (username and password) as well as the target deployment path in the yaml file. After these settings are in place, each time the main branch receives new code commits or changes on GitHub, the Pipeline will be automatically triggered. It will build and release the latest Artifact, then transfer these Artifacts to the server, achieving continuous delivery and deployment of the code. This embodies the core concept of CI/CD, which is to establish a full workflow automation from code submission, automated testing, to final product delivery and deployment through automated Pipelines.