**<PDT> I will add a description on how to set up the project permissions to allow participants to use Azure service connections, how to invite them,...**

**Ch1: Introduction**

To help you with Agile project management, Azure DevOps includes Azure Boards. With Azure Boards, you can quickly and easily start tracking user stories, backlog items, task, features, and bugs associated with your project. Please review the following introduction to Azure Boards:

1. [What is Azure Boards?](https://docs.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards)

**Challenge 1**

1. Create a work item for this and each of the remaining challenges.
2. Assign the challenge 1 work item to another member of your team, have them assign their challenge 1 work item to you.
3. Using the kanban board, move their challenge 1 work item to the resolved state.
4. After you complete each of the following challenges be sure to move the corresponding work item to the resolved state.

**Success Criteria**

1. You should have experienced creating and managing Agile project management using Azure Boards.

**Ch2: Introduction**

Historically version control has been the first component that teams have implemented, it is one of the oldest and most well understood components of DevOps. Please take a moment to review the basics of version control, specially focusing on the distributed version control technology, Git.

1. [What is version control?](https://docs.microsoft.com/en-us/azure/devops/learn/git/what-is-version-control)
2. [What is Git?](https://docs.microsoft.com/en-us/azure/devops/learn/git/what-is-git)

**Challenge 2**

Now that we have a basic understanding of version control and Git, lets get some code checked into source control. Since the language you use for development doesn’t have much of an impact on how we do DevOps we have provided you a simple ASP.NET Core C# web application to use.

1. Create an update in Azure Boards for this task as Work Item
2. There is already a Repo in the project, called “Hack3107”; this is empty for now. Your taks is to import a public GitHub Repo <https://github.com/pdtit/PDTAzDevOpsHack.git> into a new Azure Repo, named after our initials (eg. PDTHack3107).
3. Next, clone this Azure repo to your local computer ([hint](https://code.visualstudio.com/Docs/editor/versioncontrol" \l "_cloning-a-repository)).
4. Under the \Application folder, create a new readme.md file, and enter the following info:  
     
   ### Hackathon  
     
   this file has been created by <your devops account alias>
5. Commit the changes to Azure Repos
6. Create a new Branch “FridayFun” for your source code repo; link this to the first work item you created in Challenge 1.
7. Update the readme.md file in the branch, with an additional line of code “updated in this branch”, and commit the changes
8. Assign the work item to yourself, and change the status to “Doing”

**Success Criteria**

1. You should be able to go to the Azure DevOps site and under Azure Repos see your code.
2. You should be able to go to the Azure DevOps site and under Azure Boards and when you open your work item you should see your code associated to it. HINT: look in the "development" section on the work item. Why would it be important to be able to link a code change to the work item that it addresses?

**Ch3: Introduction**

Great we now have some code, now we need an environment to deploy it to. In DevOps we can automate the process of deploying the Azure Services we need with an Azure Resource Manager (ARM) template. Review the following article.

1. [Azure Resource Manager overview](https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview)
2. [Create Azure Resource Manager template](https://docs.microsoft.com/en-us/azure/azure-resource-manager/how-to-create-template)

**Challenge 3**

In Azure DevOps we can use Azure Pipelines to automate deploying our Azure infrastructure. For our application we will deploy 3 environments: Dev, Test and Prod. Each environment will consist of a Azure App Service, however all of our environments will share a single Resource Group, Azure App Service Plan, Application Insights Instance, and Azure Container Registry. NOTE: in real deployments you will likely not share all of these resources.

1. Create an update in Azure Boards for this task as Work Item
2. Create a release pipeline using the **Empty Job** template, call it Infrastructure Release
3. A release pipeline starts with an Artifact. In our pipeline we will be using the master branch of our Azure Repo.
4. Next lets create the first stage in our Infrastructure Release to deploy our ARM template to Dev. Name the stage Dev, and it should have a single Azure Resource Group Deployment task.
   1. The task will ask you what Azure Subscription, Resource Group, and Resource Group Location you wish to use.
   2. The task will also ask you what Template you want to deploy. Use the ... to pick the one in the ARM templates folder.
   3. You will need to override many of the templates parameters, replacing the <prefix> part with a unique lowercase 3-5 letter name (your initials would work fine...).
5. You should now be able to save and execute your infrastructure release pipeline successfully and see the dev environment out in Azure.
6. If everything worked, go ahead and clone the dev stage two more times for test and prod.
   1. The only change you need to make in the test and prod stages is changing the webAppName template parameter to <prefix>devops-test and <prefix>devops-prod respectively.
7. Define [petender@microsoft.com](mailto:petender@microsoft.com) as approval administrator for the prod stage.
8. You should now be able to save and execute your infrastructure release pipeline successfully and get all three environments out in Azure. Allow the approval administrator enough time to be able to approve it. He will inform you about this approval.

**Success Criteria**

1. Your infrastructure release should complete without any errors and you should see all three environments out in Azure.

NOTE: We are just scratching the surface of what is offered in Azure for Infrastructure as Code, if you are interested in learning more there is a full What the Hack focused on Azure Infrastructure as Code.

**Ch4: Introduction**

Great we now have some infrastructure and some code, lets build it. In DevOps we automate this process using something called Continuous Integration. Take a moment to review the article below to gain a better understanding of what CI is.

1. [What is Continuous Integration?](https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-integration)

**Challenge 4**

In Azure DevOps we use Azure Pipelines to automate our build process. For our application the build process will not only compile our .NET Core application, it should test it, and package it into a Docker Container and publish the container to Azure Container Registry.

1. Create an update in Azure Boards for this task as Work Item
2. Create a new YAML pipeline, this time using the **Simplcommerce.git** Repo instead. Since the app is based on .NET Core, it could potentially run on all platforms, so leave the build configured to validate all 4 (macos, linux, windows)
3. Lets kick off a build manually to ensure that we have a working build.
4. Create a new build process, this time only using the Ubuntu build agents.

**Success Criteria**

1. You understand how to create a pipeline using YAML
2. You know how to update YAML pipelines
3. Your build should complete without any errors.
4. Review the test results generated by your build. HINT: look in the logs from your build to find where the test run was published.

**Ch5: Introduction**

In DevOps after we automate our build process, we want to automate our release process, we do this with a technique called Continuous Delivery. Please take a moment to review this brief article talking about why this is important.

1. [What is Continuous Delivery?](https://docs.microsoft.com/en-us/azure/devops/learn/what-is-continuous-delivery)

**Challenge 5**

In Azure DevOps we use an Azure Pipeline to release our software. In this challenge we will deploy our container out to our dev, test, and prod environments.

1. Create an update in Azure Boards for this task as Work Item
2. Create a new Release Pipeline using the Azure App Service Deployment Template
3. To start off our deployment will only have one stage, lets call it dev
4. The output of our CI Build pipeline will be the input artifact to our CD Release pipeline, add it.
5. Enable Continuous deployment so that each time the CI pipeline finishes successfully, this pipeline will start.
6. If you look at the tasks for our dev stage you will see a single Deploy Azure App Service task, we just need to configure it.
   1. Select your subscription, a app service type of Web App for Containers (Linux), point it at our registry name pdtsimpl.azurecr.io (NOTE: here we need to fully qualify it), your image/repository name simplcommerce:latest, (NOTE: in a real-life setup, you would pull up the build number dynamically from the build process)
7. Manually kick off a release and check that your application got deployed to your dev instance.

**Success Criteria**