# Purpose

Become familiar with the dotnet CLI and cf CLI and deploy a .NET core application to Pivotal Cloud Foundry.

# Discussion points

* Directory structure
* [.NET vs .NET Core](https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server)
* .NET on Cloud Foundry
* Integrated Developer Environment (IDE) selection

# Integrated Developer Environment (IDE)

These lab instructions assume Visual Code as the IDE.

Microsoft Visual Studio can be an option if in compliance with Microsoft licensing:

* License agreement with Microsoft does not allow Pivotal Education to pre-install Visual Studio on the student machines.
* This class is classified as professional development and Microsoft licensing does not permit the use of Visual Studio Community edition.
* If students have a Visual Studio Pro or Enterprise license, they are welcome to install Visual Studio and use their license in this class. Please log out of Visual Studio at the end of each day.

At Pivotal we practice [pair programming](https://en.wikipedia.org/wiki/Pair_programming) and rotate pairs quite frequently, usually two to three times per week. Because of this, a developer may be assigned to a different workstation on any given day. To reduce friction, we adopted a standard setup and should be a class decision on the IDE that will be used.

# Project structure

1. Create a workspace directory.

Run the following to create your workspace:

mkdir -p ~/workspace

Another aspect of workstation parity is that we adopted this standard location for all project code. Similarly, the instructor team at PAL will have an easier time helping you with any problems you encounter with these labs if we can assume your code is in a standard place.

These lab instructions will assume from now on that your code is in this directory.

1. The [pal-tracker codebase](https://courses.education.pivotal.io/c/349802946/codebases/pal-tracker.zip) contains the starting points and the solutions for all the labs in this unit. Download the linked zip file and extract the codebase in the ~/workspace directory. Take some time to navigate through the tags and branches using the following command:
2. git **log** --graph --decorate --oneline --all

You will see start and solution tags for each of the coming labs.

1. [Create a repository](https://help.github.com/articles/create-a-repo/) called pal-tracker in your GitHub account. [Add this repository as a remote](https://help.github.com/articles/adding-a-remote/) called origin of your local repository. You will push all of your work to this repository during the next few labs.

Additionally, all terminal commands for the labs will assume that you are in the pal-tracker directory.

# .NET Core Application

.NET Core provides a number of templates that make it easy to start a new project.

1. View the available templates by running
2. dotnet new
3. Generate a Web API application called PalTracker. Remove the generated controller (we will add our own later).
4. dotnet new webapi --output src/PalTracker --name PalTracker
5. rm -rf src/PalTracker/Controllers
6. Generate a project for our tests called PalTrackerTests using the xunit template. Add a dependency from this project to the PalTracker project. Remove the generated test class.
7. dotnet new xunit --output **test**/PalTrackerTests --name PalTrackerTests
8. dotnet add **test**/PalTrackerTests reference src/PalTracker/PalTracker.csproj
9. rm **test**/PalTrackerTests/UnitTest1.cs
10. Create a solution file containing our two projects.
11. dotnet new sln --name PalTracker
12. dotnet sln PalTracker.sln add src/PalTracker/PalTracker.csproj
13. dotnet sln PalTracker.sln add **test**/PalTrackerTests/PalTrackerTests.csproj

# Create a controller

1. In src\PalTracker create a class called WelcomeController with the following content. We will talk more about .NET controllers in a future lab.

[Hide WelcomeController.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/dotnet-core-app/index.html" \l "pal-trackerd51c37b1-ace0-4bf6-9e37-767fce693c42)

pal-tracker/src/PalTracker/WelcomeController.cs

**using** Microsoft.AspNetCore.Mvc;

**namespace** **PalTracker**

{

[**Route("/")**]

**public** **class** **WelcomeController** : **ControllerBase**

{

[**HttpGet**]

**public** **string** **SayHello**() => "hello";

}

}

1. Run the app locally and navigate to <http://localhost:5000/> to see your message.
2. dotnet run --project src/PalTracker

By default, the built in web server, Kestrel, redirects to https on port 5001. The first time you visit the site you will need to tell the browser that it is OK to trust the self signed certificate. You do this via the Advanced option on the page. **NOTE: Accepting self signed certificates should only be done when you know the website you are visiting is the one you expect.**

1. Press Ctrl+C to shut down the application.

# CF CLI

Interaction with Pivotal Cloud Foundry is via a CLI tool called cf.

1. Verify the CLI is installed correctly by running
2. cf --version

You should see the CLI version number if you have the CLI installed:

cf version <version-number>

If the CLI is not installed, follow the [installation instructions](https://docs.cloudfoundry.org/cf-cli/install-go-cli.html) to get the CF CLI on your system.

1. When in doubt about a command, use cf --help to see what commands are available. To get help on a specific command, use cf <command> -h. For example, to get help on cf push, use cf push -h.

# CF target

The CF CLI can interact with multiple installations of PCF so we need to target a specific installation (also called a foundation). Targeting means telling our CLI about the API endpoint for a foundation.

1. Get the API endpoint for your foundation from your instructor.
2. [Log in](https://cli.cloudfoundry.org/en-US/cf/login.html) to the foundation:
3. cf login -a <cf-api-endpoint>

NOTE: cf login is not [supported in git bash](https://github.com/cloudfoundry/cli/issues/171) so use either Powershell or CMD.exe.

1. Verify the CLI is [targeted](https://cli.cloudfoundry.org/en-US/cf/target.html) correctly:
2. cf target

The output should look something like this:

API endpoint: https://api.sys.turtle.pal.pivotal.io (API version: 2.75.0)

User: pal-student@pivotal.io

Org: pal-student

Space: sandbox

# Deploy

1. Prepare the application artifacts for deployment to Pivotal Cloud Foundry.

Use dotnet CLI to publish your application dlls.

dotnet publish src/PalTracker --configuration Release

By default, the resulting artifacts can be found in pal-tracker/src/PalTracker/bin/Release/netcoreapp2.1/publish

1. Use the cf push [command](https://cli.cloudfoundry.org/en-US/cf/push.html) with the application name, a random route, and path to the artifacts (-p flag).
2. cf push pal-tracker --random-route -p src/PalTracker/bin/Release/netcoreapp2.1/publish
3. When the deployment is complete, note the url randomly generated by the CF CLI. For example:
4. requested state: started
5. instances: 1/1
6. usage: 1G x 1 instances
7. urls: pal-tracker-unjubilant-jettiness.cfapps.io <---- RANDOM URL
8. last uploaded: Mon Oct 2 15:24:38 UTC 2017
9. stack: cflinuxfs2
10. buildpack: dotnet-core
11. Confirm that the app works by navigating to http://<RANDOM URL>/.

# Push to GitHub

The application that you have built at this point is very simple, but there is now a solid foundation to build upon. Make sure to save your work by pushing your code to GitHub.

git add -AN

git add -p

git commit -m"Simple .NET Core Web API app"

git push --tags -u origin master

# Assignment submission

Submit the assignment using the **cloudNativeDeveloperSimpleApp** gradle task from within the existing assignment-submission project directory. It requires you to provide the URL of your application.

For example:

**cd** ~/workspace/assignment-submission

./gradlew cloudNativeDeveloperSimpleApp -PserverUrl=https://[app-url]

You will always submit assignments from the assignment-submission project directory going forward so keeping a separate window open for this will make it easier to submit assignments.