# Purpose

Become familiar with core deployment concepts of Cloud Foundry such as routes, scaling, environment variables, and manifests.

# Discussion points

* CF 30,000ft overview
* [CF architecture](http://htmlpreview.github.io/?https://raw.githubusercontent.com/cloudfoundry-incubator/diego-design-notes/master/clickable-diego-overview/clickable-diego-overview.html)
* [Diego talk](https://www.youtube.com/watch?v=1OkmVTFhfLY)

# 12 Factor Applications

Take a minute to read through the [12 factor](https://12factor.net/) guidelines on how to architect applications. In practice you will have to decide which ones to use and if it makes sense to adhere to all of them.

In the previous lab, you covered the first two factors by setting up your codebase in GitHub and using the dotnet CLI to explicitly declare your dependencies.

This lab will focus on the third factor: storing configuration in the environment.

There are many options for how to externalize configuration for a cloud native application. Our first choice is to use environment variables.

Another choice, externalizing configuration using a Config Server, will be introduced in an upcoming lab.

# Get started

Before starting the lab, pull in some failing tests using Git:

**cd** ~/workspace/pal-tracker

git cherry-pick externalized-config-start

Our goal is to get our test suite passing by the end of the lab.

# Environment Variables

We will use the environment variable mechanism to provide configuration to a process. In other words, it does not matter if your application is written in [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [Golang](https://en.wikipedia.org/wiki/Go_(programming_language)), or some other language, they all have the capability of reading environment variables.

Cloud Foundry can be configured to provide your application with environment variables when it is executed. In fact, it provides [several environment variables](http://docs.run.pivotal.io/devguide/deploy-apps/environment-variable.html) that your application can use to uniquely identify its execution environment.

We will refactor our application to:

* Configure a welcome controller with a message from the environment.
* Add a RESTful endpoint that returns information about our CF instance.

## Externalize Configuration

1. Create a new class called WelcomeMessage, with a Message property that is populated via constructor.

[Hide WelcomeMessage.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/externalized-configuration/index.html" \l "pal-tracker2160afc1-6569-4dc4-95d8-6131e023b1b2)

pal-tracker/src/PalTracker/WelcomeMessage.cs

**namespace** **PalTracker**

{

**public** **class** **WelcomeMessage**

{

**public** **string** Message { **get**; }

**public** **WelcomeMessage**(**string** message)

{

Message = message;

}

}

}

1. Add WelcomeMessage class to the service container and pass in the message from the configuration.
2. namespace PalTracker
3. {
4. public class Startup
5. {
6. // ...
7. public void ConfigureServices(IServiceCollection services)
8. {
9. // Add framework services.
10. services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);
11. + services.AddSingleton(sp => new WelcomeMessage(
12. + Configuration.GetValue<string>("WELCOME\_MESSAGE", "WELCOME\_MESSAGE not configured.")
13. + ));
14. }
15. // ...
16. }
17. }
18. Add a \_message backing field in the WelcomeController. Create a constructor that takes WelcomeMessage as an argument, and have the / endpoint return the welcome message.

[Show WelcomeController.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/externalized-configuration/index.html" \l "pal-tracker8b3cc24a-6d7a-41e4-8dee-dae63b5160f6)

## Verify it works

Run your app.

dotnet run --project src/PalTracker

Visit <http://localhost:5000/>. You should see an error stating that a welcome message has not yet been configured.

Now run the app and provide the welcome message as an environment variable.

dotnet run --project src/PalTracker WELCOME\_MESSAGE="hello from the environment"

Navigate again to <http://localhost:5000/> and confirm that you see hello from the environment.

## CF Environment Variables

When Cloud Foundry executes your application, it will be running with a port provided by the runtime environment. If the app has multiple instances, it will also have an instance ID set.

Cloud Foundry makes this information available to the application via environment variables. Create an endpoint to see some of that information.

Following the process introduced in the [Externalize Configuration](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/externalized-configuration/index.html#externalize-configuration) section

1. Create a class called CloudFoundryInfo to store the following environment variables:
   * **PORT**
   * **MEMORY\_LIMIT**
   * **CF\_INSTANCE\_INDEX**
   * **CF\_INSTANCE\_ADDR**

[Hide CloudFoundryInfo.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/externalized-configuration/index.html" \l "pal-tracker21588009-0eae-4207-bdf8-341017c42c3a)

pal-tracker/src/PalTracker/CloudFoundryInfo.cs

**namespace** **PalTracker**

{

**public** **class** **CloudFoundryInfo**

{

**public** **string** Port { **get**; }

**public** **string** MemoryLimit { **get**; }

**public** **string** CfInstanceIndex { **get**; }

**public** **string** CfInstanceAddr { **get**; }

**public** **CloudFoundryInfo**(**string** port, **string** memoryLimit, **string** cfInstanceIndex, **string** cfInstanceAddr)

{

Port = port;

MemoryLimit = memoryLimit;

CfInstanceIndex = cfInstanceIndex;

CfInstanceAddr = cfInstanceAddr;

}

}

}

1. Add CloudFoundryInfo to the service container and pass in the appropriate values from the configuration. (Refer to WelcomeMessage)
2. Add a test dependency on Microsoft.AspNetCore.TestHost, which we will use in our integration tests.
3. dotnet add **test**/PalTrackerTests package Microsoft.AspNetCore.TestHost --version 2.1.1
4. Create a controller called **EnvController** with a **Get** method as specified by **EnvControllerTest**. The [route](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing#attribute-routing) for the controller will be /env and method will respond to GET requests.

[Hide EnvController.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/externalized-configuration/index.html" \l "pal-tracker6eae7386-73e8-45a6-8de3-e9113f4c9fe6)

pal-tracker/src/PalTracker/EnvController.cs

**using** Microsoft.AspNetCore.Mvc;

**namespace** **PalTracker**

{

[**Route("env")**]

**public** **class** **EnvController** : **ControllerBase**

{

**private** **readonly** CloudFoundryInfo \_cloudFoundryEnv;

[**HttpGet**]

**public** CloudFoundryInfo **Get**() => \_cloudFoundryEnv;

**public** **EnvController**(CloudFoundryInfo cloudFoundryEnv)

{

\_cloudFoundryEnv = cloudFoundryEnv;

}

}

}

1. The tests must know about AspNetCore to test the controllers. Add a reference to Microsoft.AspNetCore.App to the test project.
2. dotnet add **test**/PalTrackerTests package Microsoft.AspNetCore.App --version 2.1.1
3. Run the tests ensure that everything is implemented correctly.
4. dotnet **test** **test**/PalTrackerTests

# Deploy

1. Publish the app for deployment, by running the following command.
2. dotnet publish src/PalTracker --configuration Release
3. Push the app to Cloud Foundry.
4. cf push pal-tracker -p src/PalTracker/bin/Release/netcoreapp2.1/publish
5. Notice that there is no message to display. To fix this, configure the necessary environment variables on Cloud Foundry.
6. cf **set**-env pal-tracker WELCOME\_MESSAGE "Hello from Cloud Foundry"
7. Restart your app with:
8. cf restart pal-tracker
9. Once your app is running again, reload the page and confirm that the message now appears.
10. Navigate to the **/env** endpoint and verify the actual values are displayed.

# Scaling

Cloud Foundry supports two types of scaling:

1. Vertical scaling: Each app instance has more memory or disk space.
2. Horizontal scaling: There are more app instances serving requests.

The command cf scale controls both horizontal and vertical scaling. We will demonstrate scaling using this command and view the effects on the env endpoint.

## Scale the memory

1. To see the current memory usage, run:
2. cf app pal-tracker

It will be something like '290M of 1024M'.

1. To decrease the quota, run:
2. cf scale pal-tracker -m 512M -f

Scaling memory requires restarting the container the app is running in. The **-f** flag forces the restart.

1. Wait a moment for the app to restart then navigate to the env endpoint. See that the memory quota has changed.

## Scale the number of instances

1. Change the number of instances with:
2. cf scale pal-tracker -i 2

Changing the number of instances does not require a restart.

1. Check the status of the new instance with:
2. cf app pal-tracker

If you do this fast enough, you will see the second instance switch from starting to running.

1. Wait a moment for the app to spin up new instances then navigate to the env endpoint.
2. Hit refresh on the /env endpoint a few times and see that the cfInstanceIndex eventually changes.

# Manifest

Rather than manually setting environment variables and scaling the application post-deployment, we can use a **[manifest.yml](https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html)**. This file documents requirements for the application and configures variables in the [environment](https://docs.cloudfoundry.org/devguide/deploy-apps/manifest.html#env-block). We will cover this later, but a manifest is also an appropriate place to describe the backing services that your app requires.

Create a manifest.yml file in the top-level directory (pal-tracker) like the one below:

[Hide manifest.yml](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/externalized-configuration/index.html" \l "pal-tracker96eb4b2d-2e52-42b3-812d-0fa5bafd7195)

pal-tracker/manifest.yml

---

applications:

- name: pal-tracker

path: src/PalTracker/bin/Release/netcoreapp2.1/publish/

random-route: true

env:

WELCOME\_MESSAGE: Hello **from** Cloud Foundry

To see the effect of using a manifest:

1. Delete your app with:
2. cf delete pal-tracker
3. Push your app using your new manifest with:
4. cf push
5. Visit the root endpoint and **/env** endpoint. Be aware that the URL has changed as a result of the deleting the app and random-route: true line in the manifest.
6. Make a commit and push your code to GitHub once you are sure everything is working.

# Routes

All requests to apps that are running on Cloud Foundry go through a router which holds a mapping between the route and an app. When a request comes in, it is routed to one of the app instances in a round robin fashion. We have been using --random-route and random-route: true in the class because the route an app is bound to is global to the Cloud Foundry installation. In other words, if one student takes the route pal-tracker then nobody else is able to use that route. Anyone asking to take the pal-tracker route after that would be denied.

Apps can have multiple routes bound to them which can be useful for a blue-green deployment strategy.

To get some familiarity with routing, run the following commands:

1. Map another route to your app with the map-route command, making sure to view the help for the command first. Choose a unique hostname by following [this guide](https://courses.education.pivotal.io/c/349802946/course-instructions/route-naming/index.html).
2. Navigate to both the old and new routes in a browser to check that both work and go to the same app.
3. Un-map the route you just created with the unmap-route command.

# Assignment submission

Submit the assignment using the **cloudNativeDeveloperCloudFoundry** gradle task. It requires you to provide the URL of your application. For example:

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

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