# Purpose

In this lab we will use [CircleCI](https://circleci.com/dashboard) to deploy a system of four applications to Pivotal Cloud Foundry.

# Discussion points

* Walk through system.
* Compare to app continuum.

# Get started

1. The [pal-tracker-distributed codebase](https://courses.education.pivotal.io/c/349802946/codebases/pal-tracker-distributed.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-distributed 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.

All terminal commands for the labs will assume that you are in the pal-tracker-distributed directory.

1. Create and checkout a branch for your work.
2. git checkout -b my-work
3. Push your branch to your fork of the repository.
4. git push --tags -u origin my-work

# Explore the codebase

Open the code in your IDE and take a look around.

## Directory structure

Pay special attention to the directory structure of our codebase. This codebase follows the structure outlined in the App Continuum talk.

.

├── .circleci

├── Applications

│   ├── AllocationsServer

│   ├── BacklogServer

│   ├── RegistrationServer

│   └── TimesheetsServer

├── Components

│   ├── Accounts

│   ├── AccountsTest

│   ├── Allocations

│   ├── AllocationsTest

│   ├── Backlog

│   ├── BacklogTest

│   ├── DefaultRoute

│   ├── Projects

│   ├── ProjectsTest

│   ├── TestSupport

│   ├── Timesheets

│   ├── TimesheetsTest

│   ├── Users

│   └── UsersTest

├── Databases

│   ├── allocations-database

│   ├── backlog-database

│   ├── registration-database

│   └── timesheets-database

├── IntegrationTest

├── PlatformServices

└── scripts

* .circleci folder contains artifacts pertaining to the CI pipeline of our distributed system.
* Applications folder contains our Web API applications.
* Components folder contains components of our domains and support libraries along with their respective tests.
* Databases folder contains migration information for our different schemas.
* PlatformServices folder contains local versions of the services provided by the PCF platform such a service registry and a configuration server. These will be used for local development.
* scripts folder contains build scripts used by our CI pipeline.

## Project dependencies

Inspect the project dependencies by invoking the dotnet list <PROJECT> reference command. For example:

dotnet list "./Applications/AllocationsServer/AllocationsServer.csproj" reference

dotnet list "./Applications/BacklogServer/BacklogServer.csproj" reference

dotnet list "./Applications/RegistrationServer/RegistrationServer.csproj" reference

dotnet list "./Applications/TimesheetsServer/TimesheetsServer.csproj" reference

Note that the output shows dependencies between projects, not applications.

Use this output to create a mental model of how the application and components are related. This is important to prevent circular dependencies in your codebase.

## App dependencies

The AllocationsServer, BacklogServer, and TimesheetsServer applications each make HTTP calls to the RegistrationServer application, so our application dependency graph looks like:

## Configuration

Inspect the appsettings.json files for all four microservices (in the Applications directory). Note that by default, they are all configured for running locally. On Cloud Foundry, this configuration will be overridden.

Also notice that in order Allocations, Backlog, and Timesheets microservices to interact with the Registrationmicroservice, we provide them with the **REGISTRATION\_SERVER\_ENDPOINT**.

# Explore locally

## Build and test

First, make sure that you can test and build the project locally.

1. Using the terminal, navigate to the root of the codebase.
2. Create the test and dev databases by running the **create\_databases.sql** script (in the Databases folder) against your local MySQL instance.
3. Migrate the test and development databases.

**development**

flyway -url="jdbc:mysql://localhost:3306/tracker\_allocations\_dotnet\_dev" -user=tracker\_dotnet -password=password -locations=filesystem:Databases/allocations-database migrate

flyway -url="jdbc:mysql://localhost:3306/tracker\_backlog\_dotnet\_dev" -user=tracker\_dotnet -password=password -locations=filesystem:Databases/backlog-database migrate

flyway -url="jdbc:mysql://localhost:3306/tracker\_registration\_dotnet\_dev" -user=tracker\_dotnet -password=password -locations=filesystem:Databases/registration-database migrate

flyway -url="jdbc:mysql://localhost:3306/tracker\_timesheets\_dotnet\_dev" -user=tracker\_dotnet -password=password -locations=filesystem:Databases/timesheets-database migrate

**test**

flyway -url="jdbc:mysql://localhost:3306/tracker\_allocations\_dotnet\_test" -user=tracker\_dotnet -password=password -locations=filesystem:Databases/allocations-database migrate

flyway -url="jdbc:mysql://localhost:3306/tracker\_backlog\_dotnet\_test" -user=tracker\_dotnet -password=password -locations=filesystem:Databases/backlog-database migrate

flyway -url="jdbc:mysql://localhost:3306/tracker\_registration\_dotnet\_test" -user=tracker\_dotnet -password=password -locations=filesystem:Databases/registration-database migrate

flyway -url="jdbc:mysql://localhost:3306/tracker\_timesheets\_dotnet\_test" -user=tracker\_dotnet -password=password -locations=filesystem:Databases/timesheets-database migrate

1. Run the tests to make sure everything is working properly.
2. dotnet **test**

## Run

Each in their own **separate terminal window**, start the applications as follows:

**Allocations**

dotnet run --urls "http://\*:8881" --project Applications/AllocationsServer

**Backlog**

dotnet run --urls "http://\*:8882" --project Applications/BacklogServer

**Registration**

dotnet run --urls "http://\*:8883" --project Applications/RegistrationServer

**Timesheets**

dotnet run --urls "http://\*:8884" --project Applications/TimesheetsServer

## Try it out

Exercise the curl commands below to make sure that you understand the system. Keep these commands handy, as they will be useful to test your system in future labs.

**Users**

curl -i -XPOST -H"Content-Type: application/json" localhost:8883/registration -d'{"name": "Pete"}'

curl -i localhost:8883/users/<USER-ID>

**Accounts**

curl -i localhost:8883/accounts?ownerId=<USER-ID>

**Projects**

curl -i -XPOST -H"Content-Type: application/json" localhost:8883/projects -d'{"name": "Basket Weaving", "accountId": <ACCOUNT-ID>}'

curl -i localhost:8883/projects?accountId=<ACCOUNT-ID>

**Allocations**

curl -i -XPOST -H"Content-Type: application/json" localhost:8881/allocations -d'{"projectId": <PROJECT-ID>, "userId": <USER-ID>, "firstDay": "2015-05-17", "lastDay": "2015-05-18"}'

curl -i localhost:8881/allocations?projectId=<PROJECT-ID>

**Stories**

curl -i -XPOST -H"Content-Type: application/json" localhost:8882/stories -d'{"projectId": <PROJECT-ID>, "name": "Find some reeds"}'

curl -i localhost:8882/stories?projectId=<PROJECT-ID>

**Time Entries**

curl -i -XPOST -H"Content-Type: application/json" localhost:8884/time-entries/ -d'{"projectId": <PROJECT-ID>, "userId": <USER-ID>, "date": "2015-05-17", "hours": 6}'

curl -i localhost:8884/time-entries?userId=<USER-ID>

# Set up Circle CI

Our pipeline builds all four applications and deploys to Cloud Foundry on success. The pipeline is configured to pull down the code whenever we push to GitHub. To enable this we must configure Circle CI to build our GitHub repsoitory.

1. Navigate to the [CircleCI dashboard](https://circleci.com/dashboard) and log in with your GitHub account.
2. Configure the CircleCI pipeline for your GitHub repository by going to Add Projects, clicking Set Up Project for your repository, and clicking Start building. This will start to build the master branch in your fork. Remember to set the following environment variables if they are not set:
   * CF\_API
   * CF\_ORG
   * CF\_PASSWORD
   * CF\_USERNAME

The build will run successfully, but the deployments will fail for at least two reasons:

* + There are no databases available for our applications to use.
  + Our applications' routes may conflict with those of other applications.

# Set up CF services

To help get our applications running we will now create databases and bind them to our applications on PCF.

1. Create and target a space named sandbox using the cf create-space and cf target commands.
2. Create a MySQL service instance for each application. Call them tracker-allocations-database, tracker-backlog-database, tracker-registration-database and tracker-timesheets-database respectively.

# Choose routes

The root directory contains a deployment manifest for each application in our distributed system. Familiarize yourself with a manifest, below.

[Hide manifest-allocations.yml](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-cloud-developer/deploying-distributed-systems/index.html" \l "pal-tracker-distributed7b52214c-1594-4381-b9b0-701f8ad9e7c4)

pal-tracker-distributed/manifest-allocations.yml

**---**

applications:

- name: tracker-allocations

instances: 1

memory: 256M

routes:

- route: allocations-pal-INITIALS.apps.pikes.pal.pivotal.io

path: Applications/AllocationsServer/bin/Release/netcoreapp2.1/publish

env:

REGISTRATION\_SERVER\_ENDPOINT: http://FILL\_ME\_IN/

services:

- tracker-allocations-database

Each application needs a unique route so that it does not collide with other students' applications.

1. Change the route value in each manifest to make it unique following [this guide](https://courses.education.pivotal.io/c/349802946/course-instructions/route-naming/index.html).
2. Once you have chosen routes, update the value for REGISTRATION\_SERVER\_ENDPOINT in each manifest. For example:
3. http://[registration-server-name].apps.chicken.pal.pivotal.io
4. Commit your changes and push them to Github.

This will trigger a build on CircleCI. We expect the build and deploy to succeed this time.

1. Manually trigger a few requests to the various controllers to verify that the applications are working as expected.

The Allocations, Backlog, and Timesheets applications all integrate with the Registration application, so make sure to exercise API endpoints that integrate with Registration.

# Assignment

Submit the assignment using the dotnetCloudNativeDeveloperDistributedSystemDeployment gradle task. Provide URLs of all your applications as command line arguments. For example:

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

./gradlew dotnetCloudNativeDeveloperDistributedSystemDeployment \

-PregistrationServerUrl=https://<registration-app-url> \

-PbacklogServerUrl=https://<backlog-app-url> \

-PallocationsServerUrl=https://<allocations-app-url> \

-PtimesheetsServerUrl=https://<timesheets-app-url>