# Purpose

In this lab we are going to explore remote configuration of our microservices via a [Spring Cloud Config](https://cloud.spring.io/spring-cloud-config/spring-cloud-config.html) Server. We will start with setting a simple configuration in a Git repository and integrating it in our applications.

Next we will look at more of the features of the configuration server.

# Discussion points

* Config server overview ([diagram](https://docs.pivotal.io/spring-cloud-services/common/config-server/images/config-server-fig1.png))
* Keep configuration as simple as possible.
* Use one repository per environment.

# Create a config repository

We will store the common configuration for our applications in a configuration repository.

1. Create a [new repository](https://help.github.com/articles/create-a-repo/) on GitHub called tracker-config. Make the repository public and do not initialize the repository.
2. Create a folder on your local machine to keep a local copy of this repository.
3. mkdir ~/workspace/tracker-config
4. **cd** ~/workspace/tracker-config
5. git init
6. git remote add origin git@github.com:<your-github-username>/tracker-config.git
7. Inside the tracker-config directory, create a file called application.properties with the following contents:
8. REGISTRATION\_SERVER\_ENDPOINT=http://registration-service/

This is a common configuration file that the config server provides to each of your applications.

1. Commit your changes in the tracker-config repository and push to GitHub.
2. git add application.properties
3. git commit -m"Initial commit"
4. git push -u origin master

# Configure your applications

1. Add the Steeltoe ConfigServer package to **AllocationServer**, **BacklogServer**, **TimesheetsServer**, and **RegistrationServer**projects. For example:
2. dotnet add ~/workspace/pal-tracker-distributed/Applications/AllocationsServer \
3. package Pivotal.Extensions.Configuration.ConfigServerCore --version 2.1.1
4. Remove the REGISTRATION\_SERVER\_ENDPOINT configuration from the appsettings.json of **AllocationServer**, **BacklogServer**, and **TimesheetsServer**. It will now will be loaded from the config server.
5. Update appsettings.json of **AllocationServer**, **BacklogServer**, **TimesheetsServer**, and **RegistrationServer** to:
   1. Configure the spring:application:name, which enables Config Server to return application specific configuration
   2. Set the cloud:config:failFast property, which tells our application to fail on startup if there is a problem contacting config server
   3. Provide the config server uri for local development. This uri will be overridden on Cloud Foundry.

Refer to the [Steeltoe documentation](http://steeltoe.io/docs/steeltoe-configuration/#2-2-2-configure-settings) for a full list of settings.

For example, the allocation server configuration would look like this:

{

...

+ "spring": {

+ "application": {

+ "name": "allocations-server"

+ },

+ "cloud": {

+ "config": {

+ "failFast" : true,

+ "uri": "http://localhost:8777"

+ }

+ }

+ }

}

1. Update Program.cs of the **AllocationServer**, **BacklogServer**, **TimesheetsServer**, and **RegistrationServer** to add config server as a configuration provider.

Be sure to use the Pivotal.Extensions.Configuration.ConfigServer package **instead of**Steeltoe.Extensions.Configuration.CloudFoundry.

Remove Cloud Foundry as a configuration provider, as this will be done automatically for you when using the Config server provider.

//...

- using Steeltoe.Extensions.Configuration.CloudFoundry;

+ using Pivotal.Extensions.Configuration.ConfigServer;

namespace TimesheetsServer

{

public class Program

{

//...

public static IWebHostBuilder WebHostBuilder(string[] args) =>

WebHost.CreateDefaultBuilder(args)

.UseConfiguration(new ConfigurationBuilder().AddCommandLine(args).Build())

.UseCloudFoundryHosting()

- .AddCloudFoundry()

+ .AddConfigServer()

.UseStartup<Startup>();

}

}

1. Finally, update **FlowTest.cs** to disable reliance on Config Server in the integration test.
2. public FlowTest()
3. {
4. \_registrationServer = TestAppServerBuilder()
5. .AppName("RegistrationServer")
6. .Port(8883)
7. .Database("tracker\_registration\_dotnet\_test")
8. .SetEnvironmentVariable("DISABLE\_AUTH", "true")
9. .SetEnvironmentVariable("EUREKA\_\_CLIENT\_\_SHOULDREGISTERWITHEUREKA", "false")
10. + .SetEnvironmentVariable("SPRING\_\_CLOUD\_\_CONFIG\_\_ENABLED", "false")
11. + .SetEnvironmentVariable("SPRING\_\_CLOUD\_\_CONFIG\_\_FAILFAST", "false")
12. .Build();
13. \_allocationsServer = TestAppServerBuilder()
14. .AppName("AllocationsServer")
15. .Port(8881)
16. .Database("tracker\_allocations\_dotnet\_test")
17. .SetEnvironmentVariable("REGISTRATION\_SERVER\_ENDPOINT", \_registrationServer.Url())
18. .SetEnvironmentVariable("DISABLE\_AUTH", "true")
19. .SetEnvironmentVariable("EUREKA\_\_CLIENT\_\_SHOULDFETCHREGISTRY", "false")
20. + .SetEnvironmentVariable("SPRING\_\_CLOUD\_\_CONFIG\_\_ENABLED", "false")
21. + .SetEnvironmentVariable("SPRING\_\_CLOUD\_\_CONFIG\_\_FAILFAST", "false")
22. .Build();
23. \_backlogServer = TestAppServerBuilder()
24. .AppName("BacklogServer")
25. .Port(8882)
26. .Database("tracker\_backlog\_dotnet\_test")
27. .SetEnvironmentVariable("REGISTRATION\_SERVER\_ENDPOINT", \_registrationServer.Url())
28. .SetEnvironmentVariable("DISABLE\_AUTH", "true")
29. .SetEnvironmentVariable("EUREKA\_\_CLIENT\_\_SHOULDFETCHREGISTRY", "false")
30. + .SetEnvironmentVariable("SPRING\_\_CLOUD\_\_CONFIG\_\_ENABLED", "false")
31. + .SetEnvironmentVariable("SPRING\_\_CLOUD\_\_CONFIG\_\_FAILFAST", "false")
32. .Build();
33. \_timesheetsServer = TestAppServerBuilder()
34. .AppName("TimesheetsServer")
35. .Port(8884)
36. .Database("tracker\_timesheets\_dotnet\_test")
37. .SetEnvironmentVariable("REGISTRATION\_SERVER\_ENDPOINT", \_registrationServer.Url())
38. .SetEnvironmentVariable("DISABLE\_AUTH", "true")
39. .SetEnvironmentVariable("EUREKA\_\_CLIENT\_\_SHOULDFETCHREGISTRY", "false")
40. + .SetEnvironmentVariable("SPRING\_\_CLOUD\_\_CONFIG\_\_ENABLED", "false")
41. + .SetEnvironmentVariable("SPRING\_\_CLOUD\_\_CONFIG\_\_FAILFAST", "false")
42. .Build();
43. }

# Test locally

1. Update PlatformServices/ConfigServer/src/main/resources/application.properties with the **https** (rather than the ssh) url of your config repository.
2. Start a local instance of Spring Cloud Config Server (provided in the PlatformServices directory).
3. **cd** PlatformServices
4. ./gradlew ConfigServer:bootRun
5. Start Eureka.
6. Start the applications.
7. Test your application endpoints to verify they correctly receive their configuration from Config Server and can reach the registration server.

# Deploy

1. Make a file called config-server.json which we will use when creating an instance of Config Server on Cloud Foundry. It provides the Config Server with the repository url and branch which contain our application configuration.

[Hide config-server.json](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-cloud-developer/config-server/index.html" \l "pal-tracker-distributed67326317-e95e-4e1f-a492-8d7cc669a57d)

pal-tracker-distributed/config-server.json

{

"git": {

"uri": "https://github.com/<your-github-username>/tracker-config.git",

"label": "master"

}

}

1. Create CF service:
2. cf create-service p-config-server standard tracker-config-server -c config-server.json
3. Add a service binding to tracker-config-server to each of your manifests.
4. Commit, push and wait for CircleCI to deploy.

Finally, test that your configuration is correctly loaded by making a few calls to your applications.

# Assignment

Submit the following assignment. You can find the uaaUrl, clientId, and clientSecret in the VCAP\_SERVICES of any of the microservices. Keep in mind that **clientId and clientSecret change after each deployment**.

./gradlew dotnetCloudNativeDeveloperDistributedSystemWithConfigServer \

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

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

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

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

-PuaaUrl=https://<uaa-url> \

-PclientId=<client-id> \

-PclientSecret=<client-secret>

# Cleanup

Cleanup the spaces.

cf delete-space sandbox

cf delete-space review

cf delete-space production