# Purpose

This lab introduces the Steeltoe Management library, which a number of monitoring features to an application.

# Discussion points

* Use cases for custom health check/metrics
* Alternatives: [HealthChecks](https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/implement-resilient-applications/monitor-app-health)
* Securing health check endpoints

# Get started

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

git cherry-pick health-monitoring-start

# Add Management

1. Add the Steeltoe Management package:
2. dotnet add src/PalTracker package Steeltoe.Management.CloudFoundryCore --version 2.1.0
3. Add Management endpoint services to the dependency injection container.
4. + using Steeltoe.Management.CloudFoundry;
5. namespace PalTracker
6. {
7. public class Startup
8. {
9. public void ConfigureServices(IServiceCollection services)
10. {
11. // ...
12. + services.AddCloudFoundryActuators(Configuration);
13. }
14. }
15. }
16. Configure appsettings.json to allow [Steeltoe actuators](https://steeltoe.io/docs/steeltoe-management/#1-2-9-cloud-foundry) to integrate with [Cloud Foundry Apps Manager](https://docs.pivotal.io/pivotalcf/2-0/console/using-actuators.html):
17. "ConnectionString": "Server=localhost;Database=tracker\_dotnet\_dev;Uid=tracker\_dotnet;Pwd=password;"
18. }
19. },
20. + "management": {
21. + "endpoints": {
22. + "path": "/cloudfoundryapplication"
23. + }
24. + },
25. "AllowedHosts": "\*"
26. }

## Configure the pipeline

Out of the box, Management endpoints are secured with OAuth **on Cloud Foundry**. To ease local development, Steeltoe disables this security when running locally.

Use the UseCloudFoundryActuators() extension method to add the management endpoints to the pipeline. Be sure to add it **after** the app.UseMvc() statement. This will ensure that our controllers mapping take precedence over the actuator endpoints.

+ using Steeltoe.Management.Endpoint.CloudFoundry;

namespace PalTracker

{

public class Startup

{

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

// ...

app.UseMvc();

+ app.UseCloudFoundryActuators();

}

}

}

# Management endpoints

Management exposes several [endpoints](https://steeltoe.io/docs/steeltoe-management/#1-0-management-endpoints) that show useful information about your application. Start your application and inspect the following endpoints.

* [**health**](http://localhost:5000/cloudfoundryapplication/health) - Customizable endpoint that reports application health information such as **status** (up or down) and disk space.
* [**info**](http://localhost:5000/cloudfoundryapplication/info) - Customizable endpoint that reports arbitrary application information (e.g. Git Build info, etc).
* [**loggers**](http://localhost:5000/cloudfoundryapplication/loggers) - Allows remote access and modification of logging levels in a .NET application.
* [**trace**](http://localhost:5000/cloudfoundryapplication/trace) - Reports a configurable set of trace information, for example last 100 Http requests.

# Customize the health endpoint

Custom information can be added to the **health** endpoint by creating a class that implements the **IHealthContributor** interface.

By default, the overall application health state is determined by the worst status among all the available health contributors. In other words, if there are two contributors indicating a status of up and one that is indicating a status of down, the overall status for the application will be down.

1. Create a custom health contributor called **TimeEntryHealthContributor** that:
   * Implements the **IHealthContributor** interface.
   * Reports a status of up as long as the number of time entries in the database is less than five.
   * Provides additional health details:
     + **count** - total number of time entries in the database
     + **threshold** - maximum number of time entries allowed (5)
     + **status** - string representation of the specific health status of the time entry contributor (as opposed to the entire application)
   * Returns the string "timeEntry" as the Id.

Take a look at our solution if you have trouble:

[Hide TimeEntryHealthContributor.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/health-monitoring/index.html" \l "pal-tracker3d89122d-43b1-40fb-a930-d1ae0b6c986a)

pal-tracker/src/PalTracker/TimeEntryHealthContributor.cs

**using** System.Linq;

**using** Steeltoe.Common.HealthChecks;

**using** **static** Steeltoe.Common.HealthChecks.HealthStatus;

**namespace** **PalTracker**

{

**public** **class** **TimeEntryHealthContributor** : **IHealthContributor**

{

**private** **readonly** ITimeEntryRepository \_timeEntryRepository;

**public** **const** **int** MaxTimeEntries = 5;

**public** **TimeEntryHealthContributor**(ITimeEntryRepository timeEntryRepository)

{

\_timeEntryRepository = timeEntryRepository;

}

**public** HealthCheckResult **Health**()

{

**var** count = \_timeEntryRepository.List().Count();

**var** status = count < MaxTimeEntries ? UP : DOWN;

**var** health = **new** HealthCheckResult {Status = status};

health.Details.Add("threshold", MaxTimeEntries);

health.Details.Add("count", count);

health.Details.Add("status", status.ToString());

**return** health;

}

**public** **string** Id { **get**; } = "timeEntry";

}

}

1. Register the new class with the dependency injection container with a singleton lifetime.
2. + using Steeltoe.Common.HealthChecks;
3. namespace PalTracker
4. {
5. public class Startup
6. {
7. public void ConfigureServices(IServiceCollection services)
8. {
9. // ...
10. + services.AddScoped<IHealthContributor, TimeEntryHealthContributor>();
11. }
12. }
13. }
14. Visit the [**/health**](http://localhost:5000/cloudfoundryapplication/health) endpoint again. You should now see a field for timeEntry with a status of up or down. Test the health by creating and deleting some time entries.

If you see database connection errors, ensure that your VCAP\_SERVICES environment variable is set correctly.

# Customize the info endpoint

Custom information can be added to the **info** endpoint by creating a class that implements the **IInfoContributor** interface. We will create a custom contributor that tracks whenever a time entry is created, deleted, updated, read, and listed.

1. Create an enum called **TrackedOperation**, which will represent the types of operations we want to keep track of.

[Hide TrackedOperation.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/health-monitoring/index.html" \l "pal-tracker89c3abc7-cf30-47ad-91dd-39e99e2da9d5)

pal-tracker/src/PalTracker/TrackedOperation.cs

**namespace** **PalTracker**

{

**public** **enum** TrackedOperation

{

Create,

Read,

List,

Update,

Delete

}

}

1. Create the follow **IOperationCounter** interface:

[Hide IOperationCounter.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/health-monitoring/index.html" \l "pal-trackercbc24dda-8d4c-4421-844f-bdcd93e74e5e)

pal-tracker/src/PalTracker/IOperationCounter.cs

**using** System.Collections.Generic;

**namespace** **PalTracker**

{

**public** **interface** **IOperationCounter**<**T**>

{

**void** **Increment**(TrackedOperation operation);

IDictionary<TrackedOperation, **int**> GetCounts { **get**; }

**string** Name { **get**; }

}

}

1. Update **TimeEntryControllerTest** to:
   * Create a mock of IOperationCounter<TimeEntry>.
   * Provide the mocked operation counter to TimeEntryController via constructor.
   * Verify that the counter is appropriately incremented when a TrackedOperation is performed. For example:
   * public void Read()
   * {
   * //...
   * Assert.Equal(expected, typedResponse.Value);
   * Assert.Equal(200, typedResponse.StatusCode);
   * + \_operationCounter.Verify(oc => oc.Increment(TrackedOperation.Read), Once);
   * }

Reference the solution if you need help:

[Hide TimeEntryControllerTest.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/health-monitoring/index.html" \l "pal-tracker7e54e22a-c9da-41ef-8498-ccc056dc416b)

pal-tracker/test/PalTrackerTests/TimeEntryControllerTest.cs

﻿**using** System;

**using** System.Collections.Generic;

**using** Microsoft.AspNetCore.Mvc;

**using** Moq;

**using** PalTracker;

**using** Xunit;

**using** **static** Moq.Times;

**namespace** **PalTrackerTests**

{

**public** **class** **TimeEntryControllerTest**

{

**private** **readonly** TimeEntryController \_controller;

**private** **readonly** Mock<ITimeEntryRepository> \_repository;

**private** **readonly** Mock<IOperationCounter<TimeEntry>> \_operationCounter;

**public** **TimeEntryControllerTest**()

{

\_repository = **new** Mock<ITimeEntryRepository>();

\_operationCounter = **new** Mock<IOperationCounter<TimeEntry>>();

\_controller = **new** TimeEntryController(\_repository.Object, \_operationCounter.Object);

\_operationCounter.Setup(oc => oc.Increment(It.IsAny<TrackedOperation>()));

}

[**Fact**]

**public** **void** **Read**()

{

**var** expected = **new** TimeEntry(1, 222, 333, **new** DateTime(2008, 08, 01, 12, 00, 01), 24);

\_repository.Setup(r => r.Contains(1)).Returns(true);

\_repository.Setup(r => r.Find(1)).Returns(expected);

**var** response = \_controller.Read(1);

Assert.IsType<OkObjectResult>(response);

**var** typedResponse = response **as** OkObjectResult;

Assert.Equal(expected, typedResponse.Value);

Assert.Equal(200, typedResponse.StatusCode);

\_operationCounter.Verify(oc => oc.Increment(TrackedOperation.Read), Once);

}

[**Fact**]

**public** **void** **Read\_NotFound**()

{

\_repository.Setup(r => r.Contains(1)).Returns(false);

**var** response = \_controller.Read(1);

Assert.IsType<NotFoundResult>(response);

**var** typedResponse = response **as** NotFoundResult;

Assert.Equal(404, typedResponse.StatusCode);

\_operationCounter.Verify(oc => oc.Increment(TrackedOperation.Read), Once);

}

[**Fact**]

**public** **void** **Create**()

{

**var** toCreate = **new** TimeEntry(222, 333, **new** DateTime(2008, 08, 01, 12, 00, 01), 24);

**var** expected = **new** TimeEntry(1, 222, 333, **new** DateTime(2008, 08, 01, 12, 00, 01), 24);

\_repository.Setup(r => r.Create(toCreate)).Returns(expected);

**var** response = \_controller.Create(toCreate);

Assert.IsType<CreatedAtRouteResult>(response);

**var** typedResponse = response **as** CreatedAtRouteResult;

Assert.Equal(201, typedResponse.StatusCode);

Assert.Equal("GetTimeEntry", typedResponse.RouteName);

Assert.Equal(expected, typedResponse.Value);

\_operationCounter.Verify(oc => oc.Increment(TrackedOperation.Create), Once);

}

[**Fact**]

**public** **void** **List**()

{

**var** timeEntries = **new** List<TimeEntry>

{

**new** TimeEntry(1, 222, 333, **new** DateTime(2008, 08, 01, 12, 00, 01), 24),

**new** TimeEntry(2, 999, 888, **new** DateTime(2018, 12, 05, 23, 00, 01), 8)

};

\_repository.Setup(r => r.List()).Returns(timeEntries);

**var** response = \_controller.List();

Assert.IsType<OkObjectResult>(response);

**var** typedResponse = response **as** OkObjectResult;

Assert.Equal(timeEntries, typedResponse.Value);

Assert.Equal(200, typedResponse.StatusCode);

\_operationCounter.Verify(oc => oc.Increment(TrackedOperation.List), Once);

}

[**Fact**]

**public** **void** **Update**()

{

**var** theUpdate = **new** TimeEntry(999, 888, **new** DateTime(2018, 12, 05, 23, 00, 01), 8);

**var** updated = **new** TimeEntry(1, 999, 888, **new** DateTime(2018, 12, 05, 23, 00, 01), 8);

\_repository.Setup(r => r.Update(1, theUpdate)).Returns(updated);

\_repository.Setup(r => r.Contains(1)).Returns(true);

**var** response = \_controller.Update(1, theUpdate);

Assert.IsType<OkObjectResult>(response);

**var** typedResponse = response **as** OkObjectResult;

Assert.Equal(updated, typedResponse.Value);

Assert.Equal(200, typedResponse.StatusCode);

\_operationCounter.Verify(oc => oc.Increment(TrackedOperation.Update), Once);

}

[**Fact**]

**public** **void** **Update\_NotFound**()

{

**var** theUpdate = **new** TimeEntry(999, 888, **new** DateTime(2018, 12, 05, 23, 00, 01), 8);

\_repository.Setup(r => r.Contains(1)).Returns(false);

**var** response = \_controller.Update(1, theUpdate);

Assert.IsType<NotFoundResult>(response);

**var** typedResponse = response **as** NotFoundResult;

Assert.Equal(404, typedResponse.StatusCode);

\_operationCounter.Verify(oc => oc.Increment(TrackedOperation.Update), Once);

}

[**Fact**]

**public** **void** **Delete**()

{

\_repository.Setup(r => r.Contains(1)).Returns(true);

\_repository.Setup(r => r.Delete(1));

**var** response = \_controller.Delete(1);

Assert.IsType<NoContentResult>(response);

**var** typedResponse = response **as** NoContentResult;

Assert.Equal(204, typedResponse.StatusCode);

\_operationCounter.Verify(oc => oc.Increment(TrackedOperation.Delete), Once);

}

[**Fact**]

**public** **void** **Delete\_NotFound**()

{

\_repository.Setup(r => r.Contains(1)).Returns(false);

**var** response = \_controller.Delete(1);

Assert.IsType<NotFoundResult>(response);

**var** typedResponse = response **as** NotFoundResult;

Assert.Equal(404, typedResponse.StatusCode);

\_operationCounter.Verify(oc => oc.Increment(TrackedOperation.Delete), Once);

}

}

}

1. At this point, the tests in TimeEntryControllerTest should be failing, in fact the class won't even compile. Implement the following changes to **TimeEntryController** to make them pass:
   * Receive an operation counter as a constructor argument.
   * Increment the count for each time entry operation that is performed.

See our solution if you need help:

[Hide TimeEntryController.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/health-monitoring/index.html" \l "pal-tracker8cd8a4d2-0a47-45b0-a471-6406881f11a8)

pal-tracker/src/PalTracker/TimeEntryController.cs

**using** Microsoft.AspNetCore.Mvc;

**namespace** **PalTracker**

{

[**Route("/time-entries")**]

**public** **class** **TimeEntryController** : **ControllerBase**

{

**private** **readonly** ITimeEntryRepository \_repository;

**private** **readonly** IOperationCounter<TimeEntry> \_operationCounter;

**public** **TimeEntryController**(ITimeEntryRepository repository, IOperationCounter<TimeEntry> operationCounter)

{

\_repository = repository;

\_operationCounter = operationCounter;

}

[**HttpPost**]

**public** IActionResult **Create**([FromBody] TimeEntry timeEntry)

{

\_operationCounter.Increment(TrackedOperation.Create);

**var** createdTimeEntry = \_repository.Create(timeEntry);

**return** CreatedAtRoute("GetTimeEntry", **new** {id = createdTimeEntry.Id}, createdTimeEntry);

}

[**HttpGet("{id}", Name = "GetTimeEntry")**]

**public** IActionResult **Read**(**long** id)

{

\_operationCounter.Increment(TrackedOperation.Read);

**return** \_repository.Contains(id) ? (IActionResult) Ok(\_repository.Find(id)) : NotFound();

}

[**HttpGet**]

**public** IActionResult **List**()

{

\_operationCounter.Increment(TrackedOperation.List);

**return** Ok(\_repository.List());

}

[**HttpPut("{id}")**]

**public** IActionResult **Update**(**long** id, [FromBody] TimeEntry timeEntry)

{

\_operationCounter.Increment(TrackedOperation.Update);

**return** \_repository.Contains(id) ? (IActionResult) Ok(\_repository.Update(id, timeEntry)) : NotFound();

}

[**HttpDelete("{id}")**]

**public** IActionResult **Delete**(**long** id)

{

\_operationCounter.Increment(TrackedOperation.Delete);

**if** (!\_repository.Contains(id))

{

**return** NotFound();

}

\_repository.Delete(id);

**return** NoContent();

}

}

}

1. Using OperationCounterTest as your guide, create a class called **OperationCounter** that:
   * Implements the **IOperationCounter** interface.
   * Tracks counts for each operation performed.
   * Uses the <T> type name to derive the OperationCounter.Name property:
   * **public** **string** Name => $"{**typeof**(T).Name}Operations";

Take a look at our solution if you have trouble:

[Hide OperationCounter.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/health-monitoring/index.html" \l "pal-trackerd868dea3-faf7-4486-a16b-f158c4d8df58)

pal-tracker/src/PalTracker/OperationCounter.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Collections.Immutable;

**namespace** **PalTracker**

{

**public** **class** **OperationCounter**<**T**> : **IOperationCounter**<**T**>

{

**private** **readonly** IDictionary<TrackedOperation, **int**> \_count;

**public** **OperationCounter**()

{

\_count = **new** Dictionary<TrackedOperation, **int**>();

**foreach** (**var** action **in** Enum.GetValues(**typeof**(TrackedOperation)))

{

\_count.Add((TrackedOperation) action, 0);

}

}

**public** **void** **Increment**(TrackedOperation operation)

{

\_count[operation] = ++\_count[operation];

}

**public** IDictionary<TrackedOperation, **int**> GetCounts =>

\_count.ToImmutableDictionary();

**public** **string** Name => $"{**typeof**(T).Name}Operations";

}

}

1. Register the counter with the dependency injection container using the singleton lifetime.
2. namespace PalTracker
3. {
4. public class Startup
5. {
6. public void ConfigureServices(IServiceCollection services)
7. {
8. // ...
9. + services.AddSingleton<IOperationCounter<TimeEntry>, OperationCounter<TimeEntry>>();
10. }
11. }
12. }
13. Create a class called **TimeEntryInfoContributor** that:
    * Implements the **IInfoContributor** interface.
    * Takes the time entry operation counter as a constructor argument.
    * Uses the **Contribute** method to provide the operation tallies to the info builder.

[Hide TimeEntryInfoContributor.cs](https://courses.education.pivotal.io/c/349802946/cloud-native-developer/dotnet-core-developer/health-monitoring/index.html" \l "pal-tracker104a1b7f-e23c-446d-a014-8c10475e392b)

pal-tracker/src/PalTracker/TimeEntryInfoContributor.cs

**using** Steeltoe.Management.Endpoint.Info;

**namespace** **PalTracker**

{

**public** **class** **TimeEntryInfoContributor** : **IInfoContributor**

{

**private** **readonly** IOperationCounter<TimeEntry> \_operationCounter;

**public** **TimeEntryInfoContributor**(IOperationCounter<TimeEntry> operationCounter)

{

\_operationCounter = operationCounter;

}

**public** **void** **Contribute**(IInfoBuilder builder)

{

builder.WithInfo(

\_operationCounter.Name,

\_operationCounter.GetCounts

);

}

}

}

1. Register the info contributor with the dependency injection container using the singleton lifetime.
2. + using Steeltoe.Management.Endpoint.Info;
3. namespace PalTracker
4. {
5. public class Startup
6. {
7. public void ConfigureServices(IServiceCollection services)
8. {
9. // ...
10. + services.AddSingleton<IInfoContributor, TimeEntryInfoContributor>();
11. }
12. }
13. }
14. Visit the [**/info**](http://localhost:5000/cloudfoundryapplication/info) endpoint again. You should now see the tallies for each operation performed on time entries. Test the endpoint by creating and deleting some time entries.

# Test and deploy

1. Run the tests from the test/PalTrackerTests directory to ensure that everything was implemented correctly.
2. dotnet **test** **test**/PalTrackerTests
3. Commit and push your changes. Push to production when you are satisfied with the result and confirm the integration with [Cloud Foundry Apps Manager](https://docs.pivotal.io/pivotalcf/2-0/console/using-actuators.html).

# Assignment submission

Submit the assignment using the **cloudNativeDeveloperHealthMonitoring** gradle task. It requires that you disable security on the management endpoints and provide the URL of your staging application as follows:

cf **set**-env pal-tracker MANAGEMENT\_\_ENDPOINTS\_\_CLOUDFOUNDRY\_\_ENABLED false

cf restart pal-tracker

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

./gradlew cloudNativeDeveloperHealthMonitoring -PserverUrl=https://[staging-app-url]/cloudfoundryapplication

Do not forget to set the proper environment variables on your staging application to turn off security for the assignment submission.

# Extra

If you are finished with this assignment before the rest of the class is done, try implementing [dynamic logging](https://steeltoe.io/docs/steeltoe-management/#1-2-5-loggers) in Steeltoe so that it integrates with [Apps Manager](https://docs.pivotal.io/pivotalcf/2-0/console/using-actuators.html#manage-log-levels).