Pivotal

Cloud Native Patterns & Services

NETFLIX OSS & CLOUD SERVICES

Platform Architecture Team

amazon.ca Try Prime

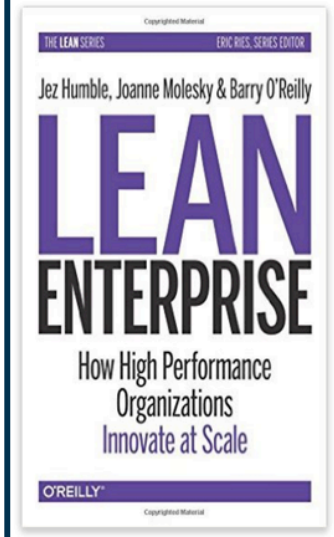
All | lean enterprise

Back to University Presented by Scotiabank

Shop by Department | Nenad's Store | Deals Store | Gift Cards | Sell | Help | EN | Hello, Nenad Your Account | Try Prime | Wish List | 0 Cart

Books | Advanced Search | Browse Subjects | New Releases | Best Sellers | Globe and Mail Best Sellers | Best Books of the Month | Children's Books | Textbooks | Today's Deals | Livres en français

‹ Back to search results for "lean enterprise"



See all 2 images

Lean Enterprise: How High Performance Organizations Innovate at Scale Hardcover –

Jan 3 2015

by Jez Humble (Author), Joanne Molesky (Author), Barry O'Reilly (Author)

★★★★★ 2 customer reviews

See all 4 formats and editions

Kindle Edition	Hardcover	Paperback
CDN$ 10.22	CDN$ 28.08	from CDN$ 165.76

Read with Our Free App | 8 Used from CDN$ 14.87, 33 New from CDN$ 16.32 | 1 Used from CDN$ 165.76, 1 New from CDN$ 207.20

Back to University 2016
Save on College Prep Essentials on Amazon.ca. Shop now

How well does your organization respond to changing market conditions, customer needs, and emerging technologies when building software-based products? This practical guide presents Lean and Agile principles and patterns to help you move fast at scale—and demonstrates why and how to apply these methodologies throughout your organization, rather than with just one department or team.

Through case studies, you'll learn how successful enterprises have rethought everything from

Read more

Vous voulez voir cette page en français ? Cliquez ici.

Share

CDN$ 28.08
List Price: ~~CDN$ 32.40~~
You Save: CDN$ 4.32 (13%)

FREE Shipping

In Stock.
Ships from and sold by Amazon.ca.
Gift-wrap available.

Quantity: 1

Yes, I want FREE Two-Day Shipping with Amazon Prime

Add to Cart

Turn on 1-Click ordering for this browser

Want it Tuesday, September 20? Order it in the next **2 hours and 6 minutes** and choose **One-Day Shipping** at checkout.

Special Offers and Product Promotions

Cloud Native Architectures-Light Side

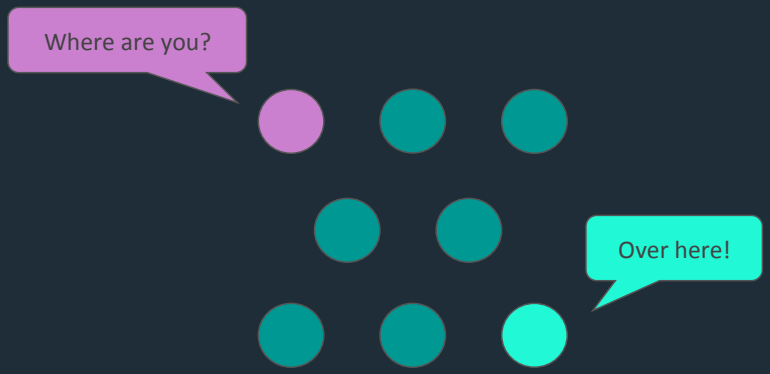
Light Side of the Cloud

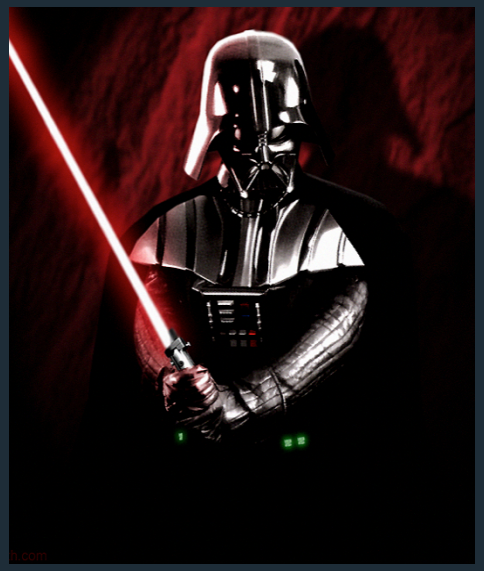
- Scalability
- High Availability
- Velocity: Continuous Delivery
- On-Demand Provisioning

Pivotal

Finding Services

Dark Side of the Cloud: Finding Services

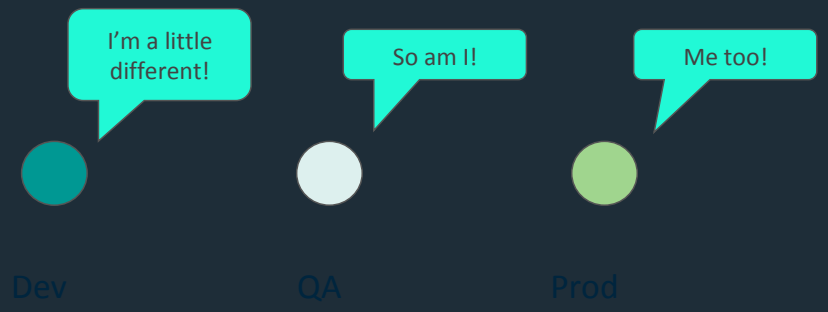


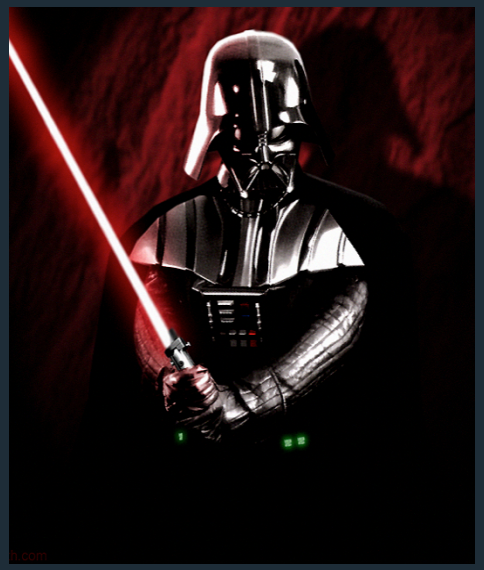


Pivotal

Managing Configuration Differences

Dark Side of the Cloud: Managing Configuration Differences

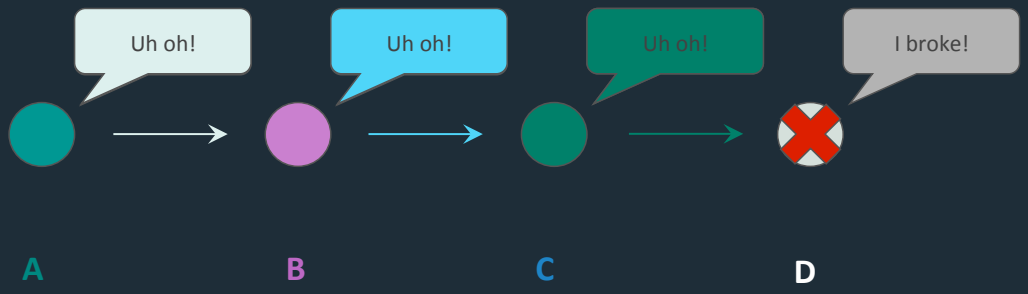


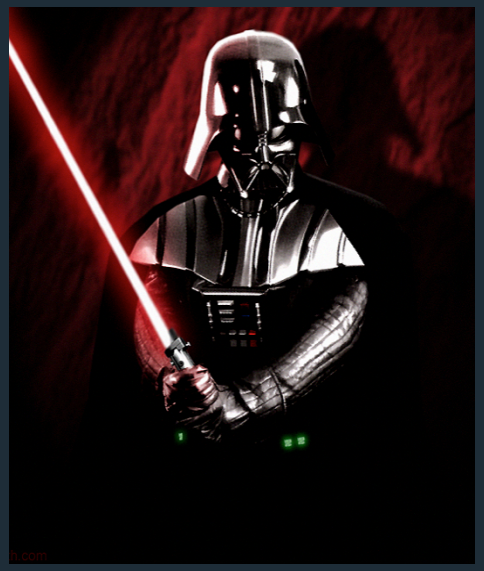


Pivotal

Handling Failure

Dark Side of the Cloud: Handling Failure

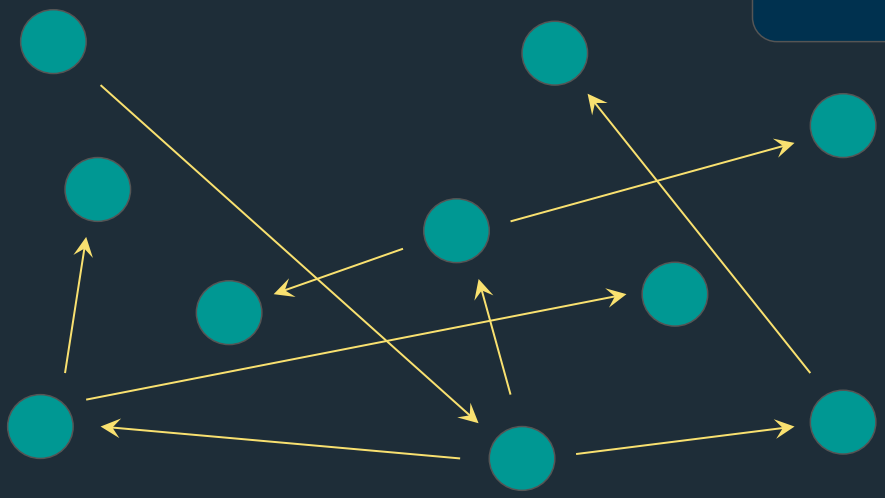


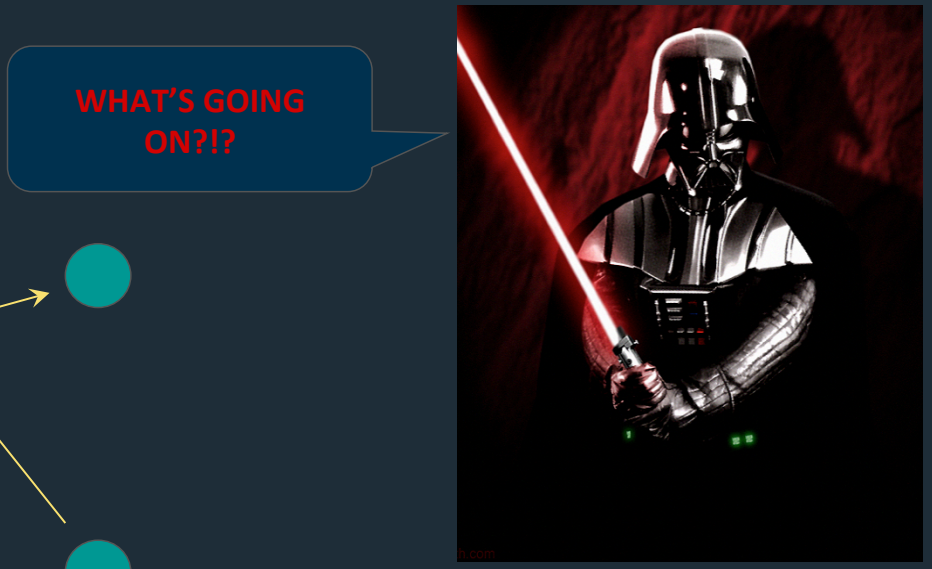


Pivotal

Following Call Graphs

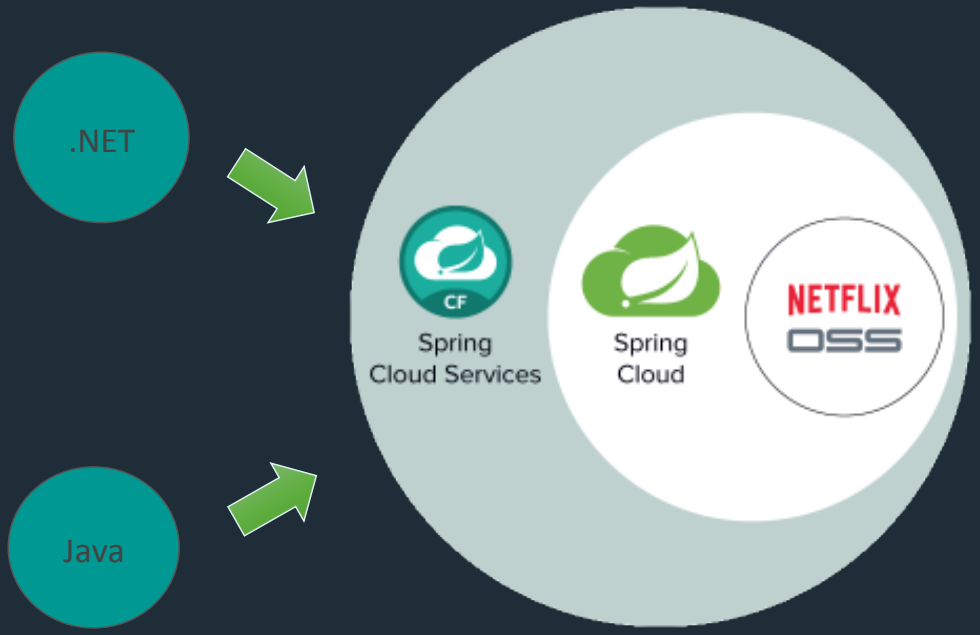
Dark Side of the Cloud: Following call graphs





Pivotal

Cloud Native Patterns & Services Evolution



Services Marketplace

Circuit Breaker
Circuit Breaker Dashboard for Spring Cloud Applications

Config Server
Config Server for Spring Cloud Applications

Service Registry
Service Registry for Spring Cloud Applications

Pivotal

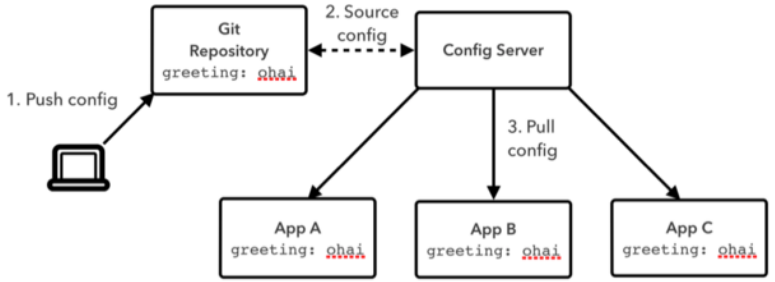
Other Contributors Cloud Infrastructure Libraries

- Twitter, Facebook and Hashicorp all have open-sourced other cloud infrastructure libraries
  - Zipkin
  - Consul
- Complementary and competing solutions
  - Form a bazaar of ideas and solutions

Todo: add MS here

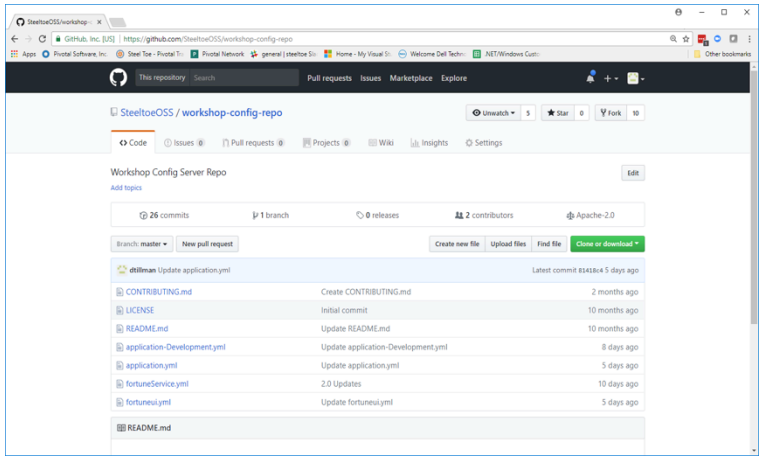
Cloud Native Service - Config Server

- Enables centralizing application configurations
- Supports different back end storage options
  - Git repositories
  - File System
  - Hashicorp Vault
  - Composite
- Configuration file format support
  - Java properties (.properties)
  - YAML (.yml)
- Exposes configurations via REST based API, http://localhost:8080/
  - Data returned in JSON
- Client pulls configuration providing
  - {AppName}
  - {Profile} – one or more
  - {Label} – zero or more, meaning varies by back end
- Cloud Foundry version of Config server installed with Spring Cloud Services (SCS)
  - Version is protected by OAuth 2.0 tokens
  - Uses TLS/SSL connections



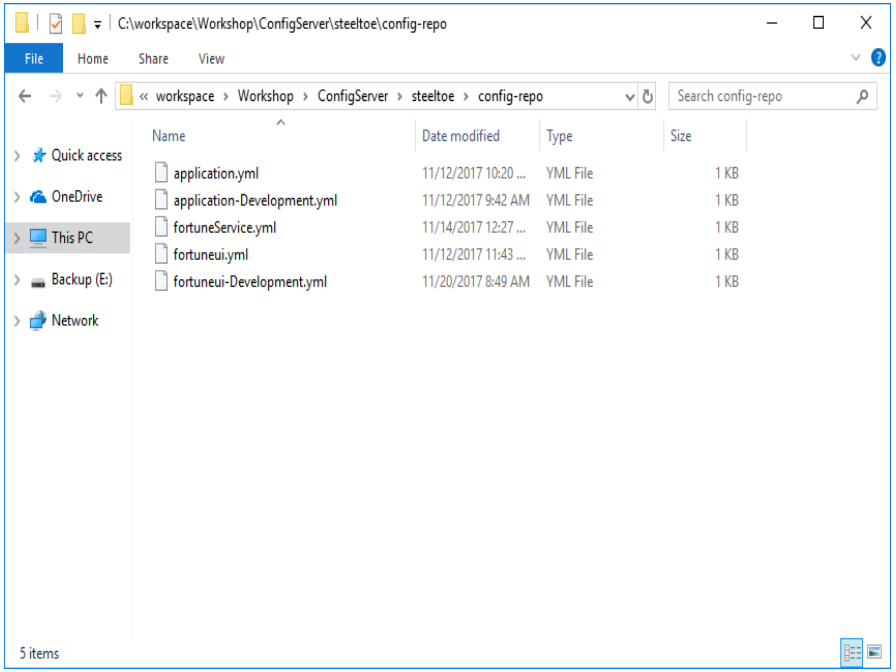
Config Server – Git Backend

- Client request parameter mappings
  - {AppName} -> file name pattern in Git repo
  - {Profile} -> file name pattern in Git repo
  - {Label} -> Git branch, tag, commit-id, defaults to ‘master’
- Selects configuration files from Git repo with patterns
  - /application.{suffix}
  - /{AppName}.{suffix}
  - /application-{Profile}.{suffix}
  - /{AppName}-{Profile}.{suffix}
- Selected files returned in a list with precedence
  - Above shows list order - least to highest precedence
  - Values in higher precedence files override lower
- Example Config Server requests
  - e.g. `curl localhost:8888/{AppName}/{Profile}`
  - e.g. `curl localhost:8888/{AppName}/{Profile1},{Profile2}`
  - e.g. `curl localhost:8888/{AppName}/{Profile}/{Label}`



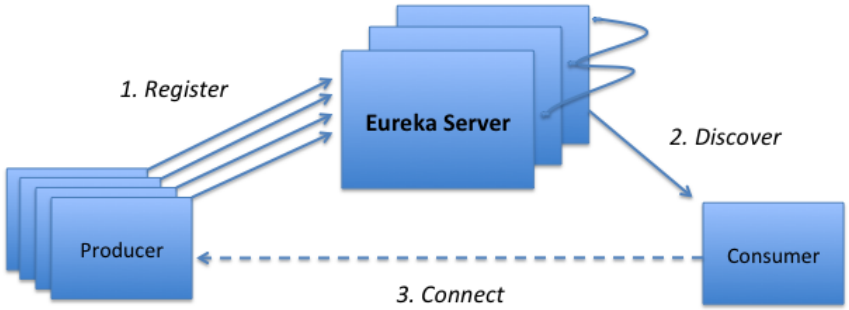
Config Server – File System Backend

- Client request parameter mappings
  - {AppName} -> file name pattern in file system
  - {Profile} -> file name pattern in file system
  - {Label} -> directory name pattern in file system
- Selects configuration files from file system with patterns
  - /application.{suffix}
  - /{AppName}.{suffix}
  - /{Label}/application.{suffix}
  - /{Label}/{AppName}.{suffix}
  - /application-{Profile}.{suffix}
  - /{AppName}-{Profile}.{suffix}
  - /{Label}/application-{Profile}.{suffix}
  - /{Label}/{AppName}-{Profile}.{suffix}
- Selected files returned as a list with precedence
  - Above shows list order - least to highest precedence
  - Values in higher precedence files override lower
- Example Config Server requests
  - e.g. `curl localhost:8888/{AppName}/{Profile}`
  - e.g. `curl localhost:8888/{AppName}/{Profile}/{Label}`



Cloud Native Service - Eureka Service Registry

- Server for registering and discovering services
  - Exposes a REST based API http://localhost:8761/eureka
- Applications register addresses by name with Eureka Server
  - Heartbeats renew leases (30 seconds)
  - Three strikes, you're out (90 seconds)
  - Registrations optionally replicated to peers
- Eureka Client pulls entire service registry
  - Cached, used to lookup services by name
- Resilience
  - Multiple Eureka Servers
  - Clients cache copy of registry



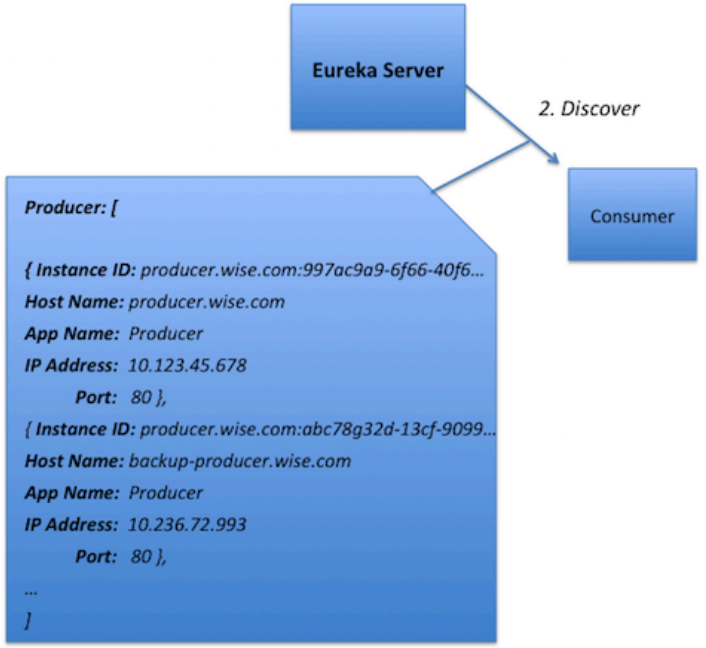
- Cloud Foundry version of Eureka server installed with Spring Cloud Services (SCS)
  - Version is protected by OAuth 2.0 tokens
  - Uses TLS/SSL connections

Eureka Server – Service Registration

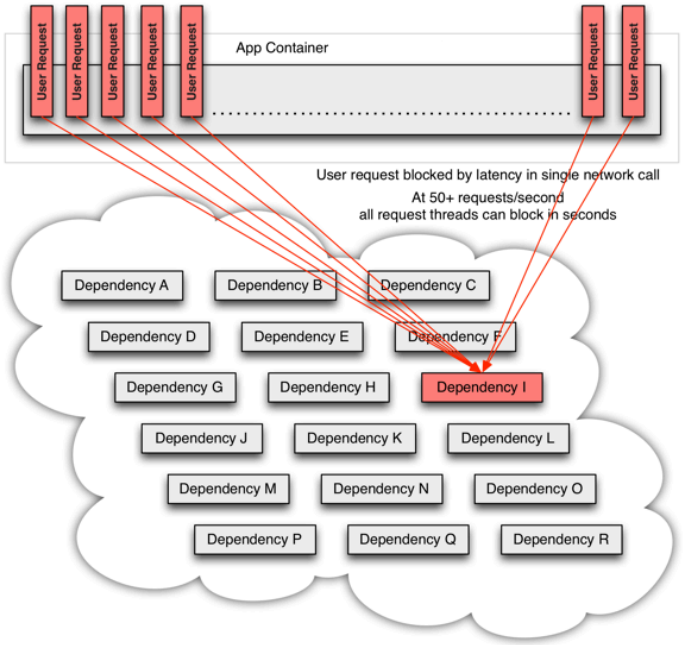
- Application instances register with Eureka Server providing
  - Application Name
  - Host Name
  - Port
  - Instance ID
  - IP Address
  - Other metadata
- Default is to register by Host Name
  - Optionally register by IP Address

Eureka Server – Service Discovery

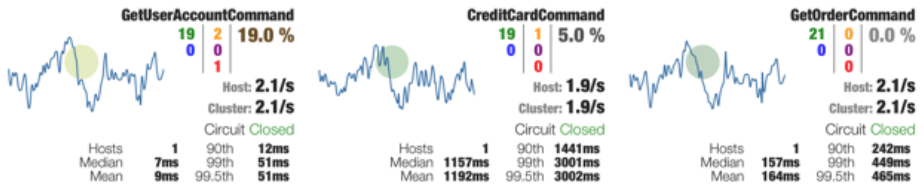
- Application instances periodically fetch the registry
  - Default every 30 second
  - Cached locally
  - Deltas retrieved after full fetch
- Application uses cache to lookup instances of other applications
  - Client load balancing possible



Cloud Native Service - Hystrix

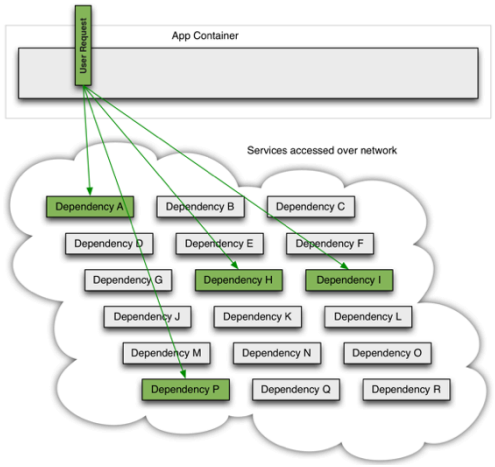


- Hystrix consists of two parts
  - Fault tolerance framework for dealing with failures and latency in distributed systems
    - Isolation patterns to limit impacts
  - Near real-time monitoring & alerting
    - Shorten time-to-discover and time-to-recover from failures

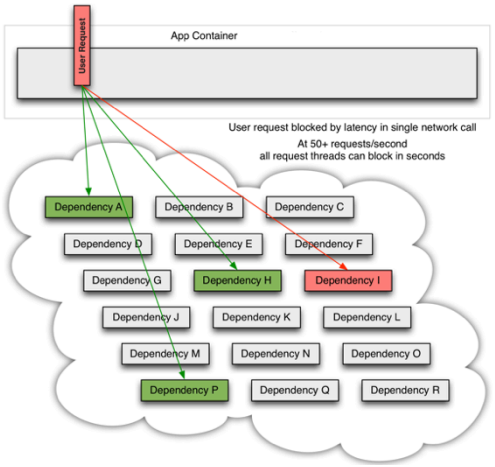


Hystrix – Fault Tolerance Framework

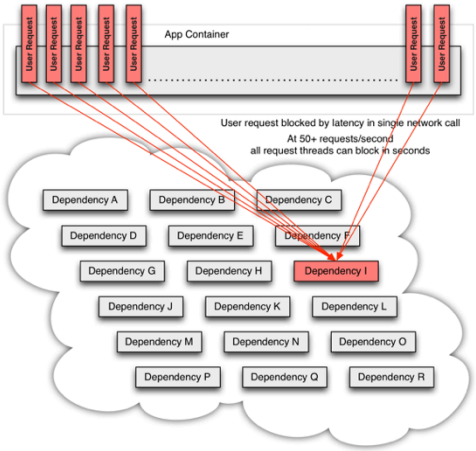
Healthy System



Backend Dependency Becomes Latent

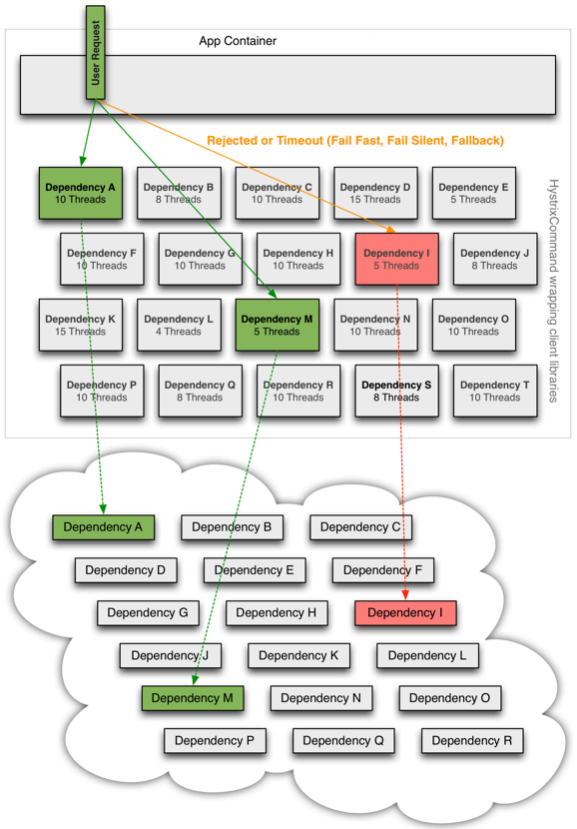


Latency cascades resources become saturated



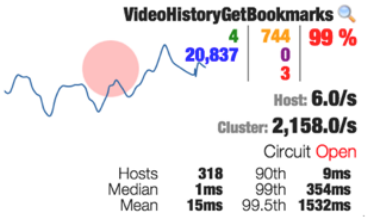
Hystrix

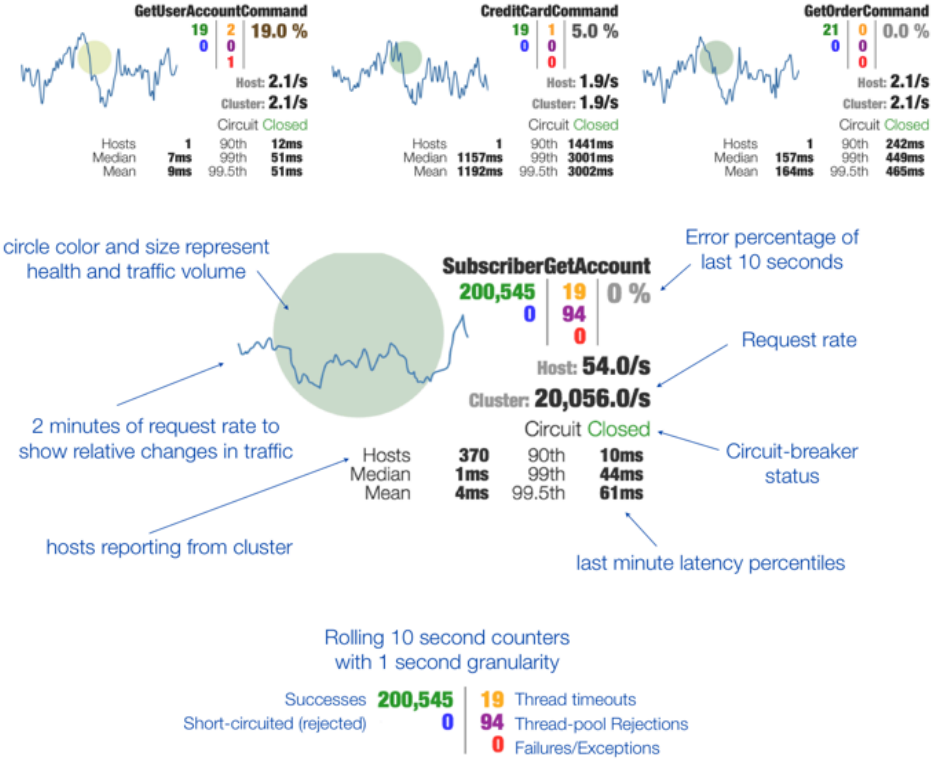
- Fault tolerance framework
  - Wrap all external dependencies in Hystrix commands
    - Executed on separate thread
    - Execute fallback logic on failures
  - Configurable thread pools for each dependency
    - Fail fast if pool consumed
  - Measures success, failures, timeouts, thread rejections, etc.
  - Trip circuit-breaker to stop requests to failed dependency
    - Periodically check
  - Gather metrics and report back to a dashboard for monitoring



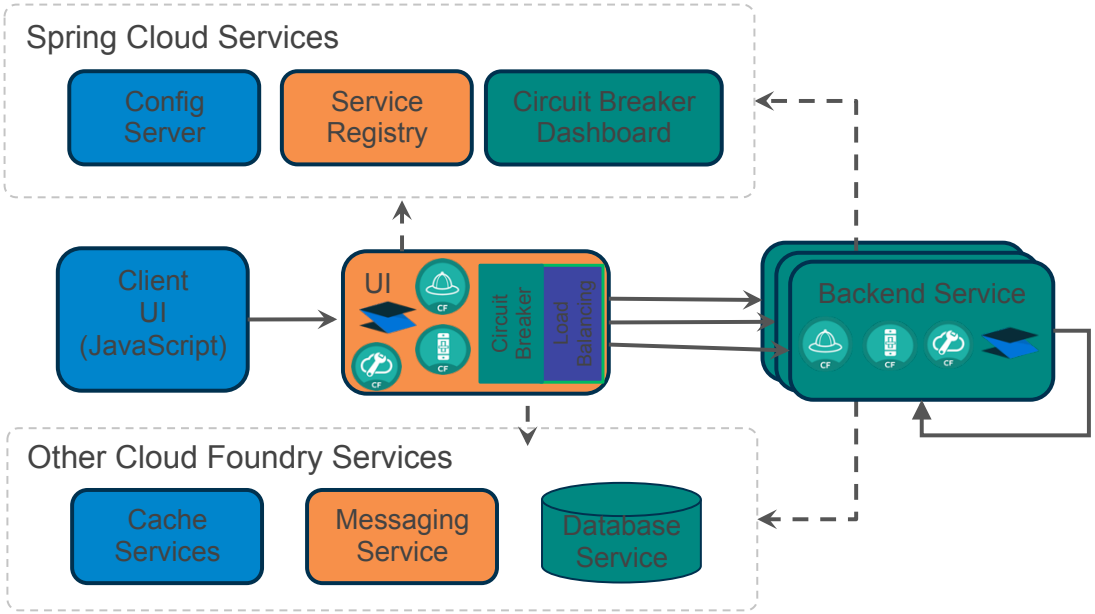
Hystrix – Monitoring and Alerting

- Near real-time monitoring & alerting
  - Circuit breaker status
  - Thread pool usage
  - Request rates
  - Error percentages

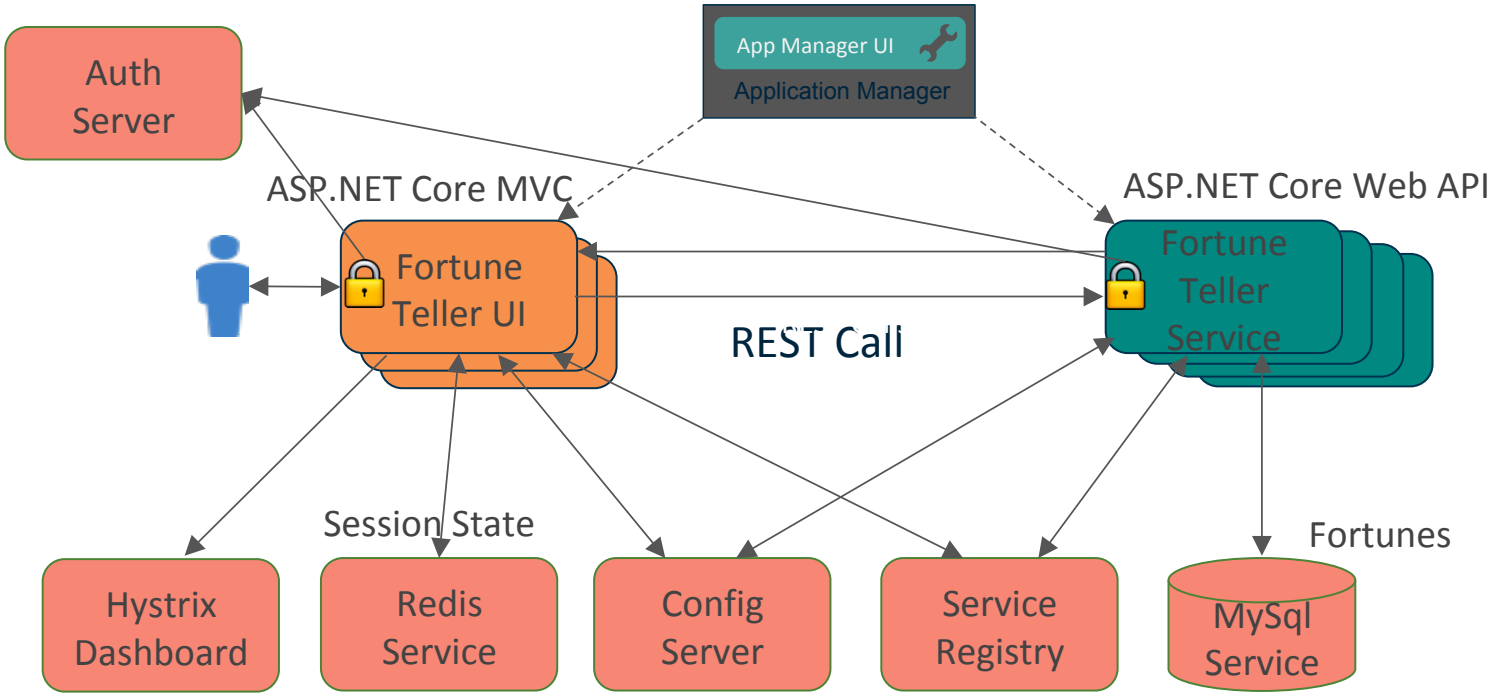




Putting it all together
Application Architecture using Cloud Native Services



Fortune Teller Demo



Steeltoe

.NET, SPRING CLOUD, SPRING CLOUD SERVICES, CONFIGURATION, EUREKA, CONNECTORS, HYSTRIX, MANAGEMENT

Steeltoe Project

Facilitates Twelve-Factor Contract on .NET

http://steeltoe.io

Enabling Cloud Native Applications on .NET

- Simplifies using .NET & ASP.NET on Cloud Foundry
  - Connectors (e.g. MySql, Redis, Postgres, RabbitMQ, OAuth, etc.)
  - Security providers (e.g. OAuth SSO, JWT, Redis KeyRing Storage, etc.)
  - Configuration providers (e.g. Cloud Foundry)
  - Management & Monitoring
- Simplifies using Spring Cloud Services
  - Configuration server provider (e.g. Config Server, etc.)
  - Service Discovery (e.g. Eureka, etc.)
  - Circuit Breaker (e.g. Netflix Hystrix)
  - Distributed Tracing (e.g. Slueth coming)

Steeltoe Frameworks

- Frameworks are Open Source
  - https://github.com/SteeltoeOSS
  - Functionally organized (Configuration, Discovery, Connectors, etc.)
- .NET support
  - .NET Core (Windows, Linux & OSX)
  - .NET Framework
- Application type support
  - ASP.NET 4 - MVC, WebForm, WebAPI, WCF
  - ASP.NET Core (.NET Framework and .NET Core)
  - Console apps (.NET Framework and .NET Core)

Steeltoe Frameworks cont'd

- Configuration – additional .NET Configuration providers
  - CloudFoundry – parse VCAP_*, CF_* and add to apps configuration
  - Config Server Client – access to Spring Cloud Config Server
- Connectors – simplifies configuring and injecting the following as services
  - MySql, MySql EF6, MySql EFCore – Connections and DbContexts
  - Postgres, Postgres EFCore – Connections and DbContexts
  - SqlServer, SqlServer EF6, SqlServer EFCore – Connections and DbContexts
  - RabbitMQ – Connection Factory
  - Redis – Microsoft IDistributedCache & StackExchange IConnectionMultiplexor
  - OAuth – access connection details from CF UAA & Pivotal SSO Service
- Circuit Breaker
  - Hystrix – Netflix based, latency and fault tolerance library

Steeltoe Frameworks cont'd

- Discovery – service registry clients
  - Netflix Eureka Client – registration and discovery via Eureka Server
- Security – providers for Cloud Foundry and ASP.NET Core security integration
  - OAuth2 provider – Cloud Foundry integration with UAA/Pivotal SSO
  - JWT provider - Cloud Foundry integration with UAA/Pivotal SSO
  - Redis DataProtection KeyStorage connector – use Cloud Foundry Redis service for key ring storage
- Management – production monitoring and management
  - Health
  - Build info – e.g. Git, etc.
  - Dynamically adjustable log levels
  - Request/Response Traces

Steeltoe NuGets

- NuGet feeds
  - Development: https://www.myget.org/gallery/steeltoedev
  - Stable: https://www.myget.org/gallery/steeltoemaster
  - Release & Release Candidates: https://www.nuget.org/
- NuGet naming conventions
  - Steeltoe.X.Y.Z – core X.Y.Z functionality, application type independent
    - e.g. Steeltoe.Extensions.Configuration.CloudFoundry
  - Steeltoe.X.Y.ZCore – ASP.NET Core DI support
    - e.g. Steeltoe.Extensions.Configuration.CloudFoundryCore
  - Steeltoe.X.Y.ZAutofac – ASP.NET 4 Autofac support
    - e.g. Steeltoe.Extensions.Configuration.CloudFoundryAutofac
  - Other DI container support over time

Steeltoe Help

- Documentation – http://steeltoe.io
- Slack - https://slack.steeltoe.io/
- Samples – https://github.com/SteeltoeOSS/Samples
  - Functionally - organized by framework area (e.g. Configuration, Discovery, etc.)
    - ASP.NET 4 and ASP.NET Core samples
  - Multi-functional – illustrates using several Steeltoe components
    - MusicStore – micro-services app built from the ASP.NET Core reference app
    - FreddysBBQ - a polyglot (i.e. Java and .NET) micro-services based sample app
    - Workshop – Fortune Teller using all of the Steeltoe components

Steeltoe Management

ASP.NET CORE, MANAGEMENT, ENDPOINTS, TRACE, LOGS,

Steeltoe Management Overview

- A set of frameworks you can use to add monitoring and management features to your application
  - Typically expose management features as HTTP REST endpoints using management actuators
    - Seamlessly integrates with Pivotal Apps Manager
  - Can expose endpoints using any technology
- Several Endpoints
  - Health
  - App Info
  - Loggers
  - Trace
  - Cloud Foundry
- Application type support
  - ASP.NET Core (.NET Framework and .NET Core)

Steeltoe Health Endpoint

- Used to check the health of an application
  - Health is collected from all `IHealthContributor`
  - Final application health is computed by an `IHealthAggregator`
- Usage
  - Add NuGet reference to project
  - Create any needed custom `IHealthContributor`
  - Add all `IHealthContributor` to service container
  - Add Health actuator to service container and pipeline

Health Endpoint – Add NuGet References

- Health Endpoint
  - Basic functionality - `Steeltoe.Management.Endpoint`
  - ASP.NET Core Actuator - `Steeltoe.Management.EndpointCore`
  - All ASP.NET Core Actuators - `Steeltoe.Management.CloudFoundryCore`

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
    <PackageReference Include="Steeltoe.Management.CloudFoundryCore" Version= "x.y.z" />
<ItemGroup>

</Project>
```

Health Endpoint – Create Custom HealthContributor

- Define class which derives from `IHealthContributor`
  - Give it an ID
- Create `Health()` method to calculate returned `Health`
  - Result put in `Health.Status`
  - Provide more health information in `Health.Details`
    - `status`
    - `error`

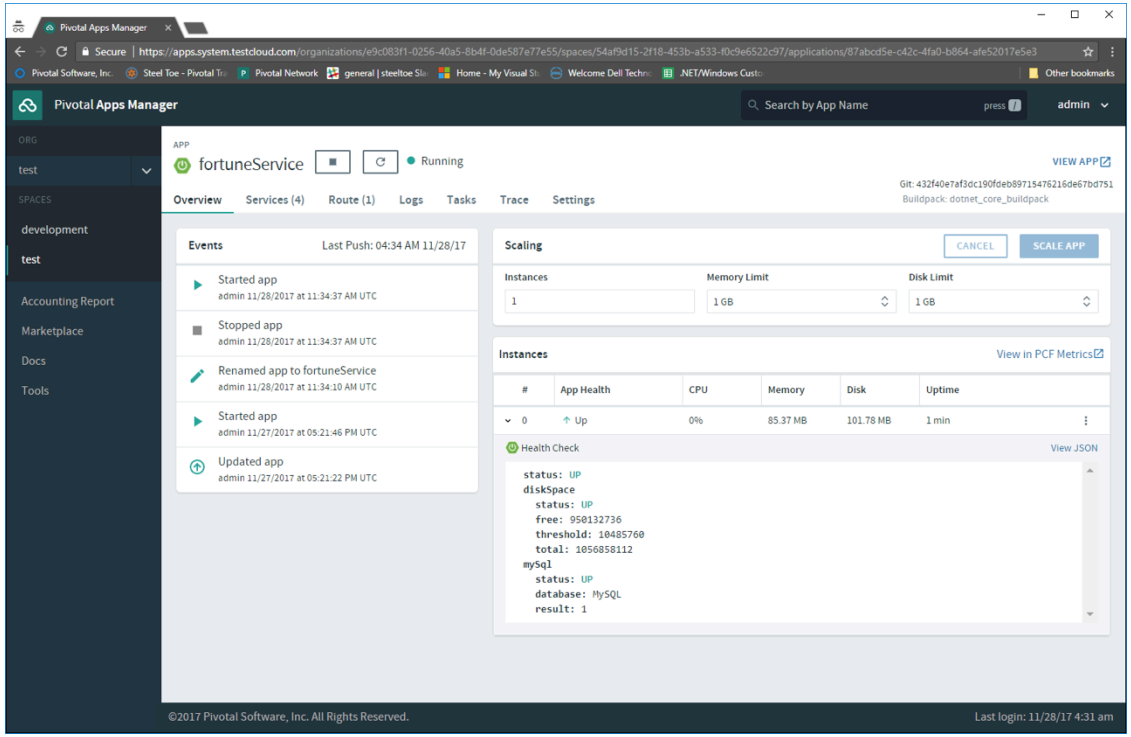
```
public class MySqlHealthContributor : IHealthContributor
{
  FortuneContext _context;
  public MySqlHealthContributor(FortuneContext dbContext)
  {
    _context = dbContext;
  }
  public string Id { get; } = "mySql";
  public Health Health()
  {
    Health result = new Health();
    result.Details.Add("database", "MySQL");
    MySqlConnection _connection = null;
    try {
       _connection = _context.Database.GetDbConnection() as MySqlConnection;
       if (_connection != null) {
         _connection.Open();
         MySqlCommand cmd = new MySqlCommand("SELECT 1;", _connection);
         var qresult = cmd.ExecuteScalar();
         result.Details.Add("result", qresult);
         result.Details.Add("status", HealthStatus.UP.ToString());
         result.Status = HealthStatus.UP;
       }
    } catch (Exception e) {
       result.Details.Add("error", e.GetType().Name + ": " + e.Message);
       result.Details.Add("status", HealthStatus.DOWN.ToString());
       result.Status = HealthStatus.DOWN;
    } finally { if (_connection != null) _connection.Close(); }
    return result;
  }
}
```

Health Endpoint – Add Health Actuator

- Add any custom `IHealthContributor` to container
- Add Health endpoint actuator or all actuators
  - `AddHealthActuator`
  - `AddCloudFoundryActuators`
- Add Health endpoint middleware to request pipeline
  - `UseHealthActuator`
  - `UseCloudFoundryActuators`

```
public class Startup
{
  public IConfiguration Configuration { get; }

  public void ConfigureServices(IServiceCollection services)
  {
    services.AddOptions();

    services.AddScoped<IHealthContributor, MySqlHealthContributor>();

    // services.AddHealthActuator(Configuration);
    services.AddCloudFoundryActuators(Configuration);
}

  public void Configure(IApplicationBuilder app, IHostingEnvironment env)
  {
    .....

    // app.UseHealthActuator();
    app.UseCloudFoundryActuators();

    app.UseMvc();

}

}
```

Health Endpoint - Using on Cloud Foundry



Steeltoe App Info Endpoint

- Used to gather general application information
  - Info is collected from all `IInfoContributor`
    - `GitInfoContributor` – returns `git.properties` file
    - `AppSettingsInfoContributor` – returns appsettings info
- Usage
  - Add NuGet references to project
  - Use `GitInfo` in project to create `git.properties`
  - Add Info actuator to service container and pipeline

App Info Endpoint – Add NuGet References

- App Info Endpoint
  - Basic functionality - `Steeltoe.Management.Endpoint`
  - ASP.NET Core Actuator - `Steeltoe.Management.EndpointCore`
  - All ASP.NET Core Actuators - `Steeltoe.Management.CloudFoundryCore`
- Use GitInfo to provide Git information during MSBuild

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
    <PackageReference Include="Steeltoe.Management.CloudFoundryCore" Version= "x.y.z" />
    <PackageReference Include="GitInfo" Version="2.0.1" />
<ItemGroup>

</Project>
```

App Info Endpoint – Using GitInfo

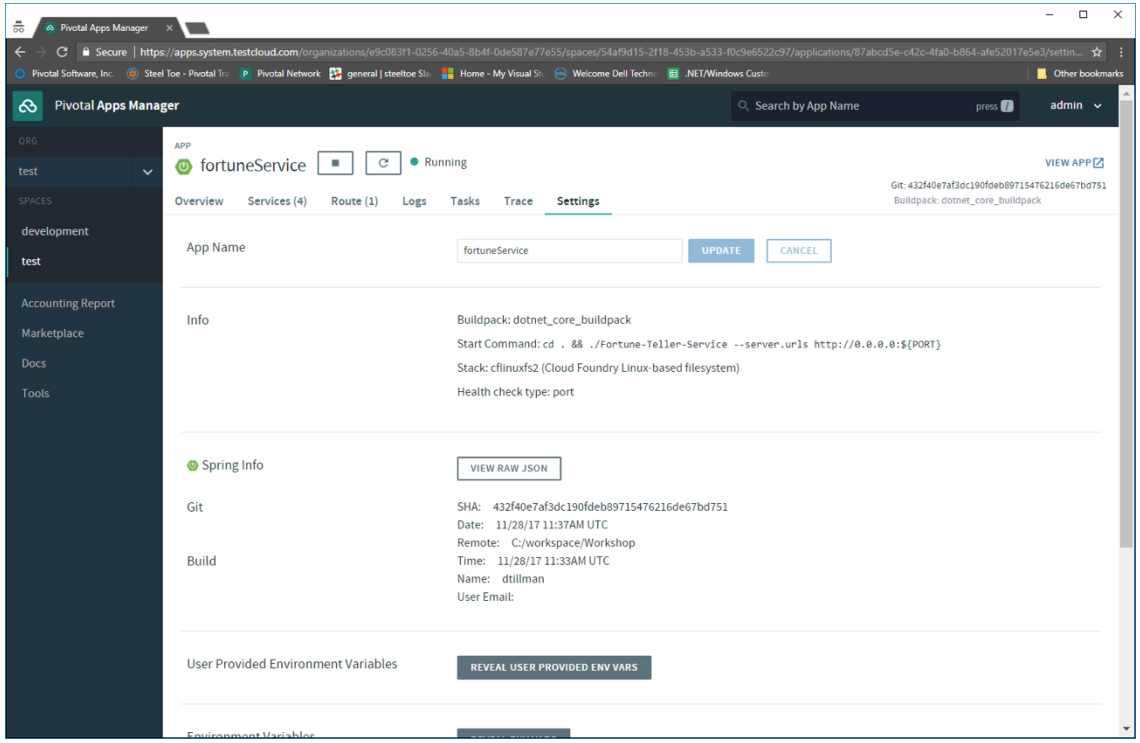
```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
    <PackageReference Include="Steeltoe.Management.CloudFoundryCore" Version= "x.y.z" />
    <PackageReference Include="GitInfo" Version= "x.y.z" />
    <None Include="git.properties">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
    </None>
<ItemGroup>
 <Target Name="_GitProperties" AfterTargets="CoreCompile">
    <WriteLinesToFile File="git.properties" Lines="git.remote.origin.url=$(GitRoot)" Overwrite="true" />
    <WriteLinesToFile File="git.properties" Lines="git.build.version=$(GitBaseVersion)" Overwrite="false" />
    <WriteLinesToFile File="git.properties" Lines="git.commit.id.abbrev=$(GitCommit)" Overwrite="false" />
    <WriteLinesToFile File="git.properties" Lines="git.commit.id=$(GitSha)" Overwrite="false" />
    <WriteLinesToFile File="git.properties" Lines="git.tags=$(GitTag)" Overwrite="false" />
    <WriteLinesToFile File="git.properties" Lines="git.branch=$(GitBranch)" Overwrite="false" />
    <WriteLinesToFile File="git.properties" Lines="git.build.time=$([System.DateTime]::Now.ToString('O'))" Overwrite="false" />
    <WriteLinesToFile File="git.properties" Lines="git.build.user.name=$([System.Environment]::GetEnvironmentVariable('USERNAME'))" Overwrite="false" />
    <WriteLinesToFile File="git.properties" Lines="git.build.host=$([System.Environment]::GetEnvironmentVariable('COMPUTERNAME'))" Overwrite="false" />
  </Target>
</Project>
```

App Info Endpoint – Add Info Actuator

- Add any custom `IInfoContributor` to container
- Add Info endpoint actuator or all actuators
  - `AddInfoActuator`
  - `AddCloudFoundryActuators`
- Add Info endpoint middleware to request pipeline
  - `UseInfoActuator`
  - `UseCloudFoundryActuators`

```
public class Startup
{
  public IConfiguration Configuration { get; }

  public void ConfigureServices(IServiceCollection services)
  {
    services.AddOptions();

    // services.AddInfoActuator(Configuration);
    services.AddCloudFoundryActuators(Configuration);
}

  public void Configure(IApplicationBuilder app, IHostingEnvironment env)
  {
    .....

    // app.UseInfoActuator();
    app.UseCloudFoundryActuators();

    app.UseMvc();

}

}
```

App Info Endpoint - Using on Cloud Foundry



Steeltoe Loggers Endpoint

- Used to view and configure the active loggers and their logging levels at runtime
  - Must use the Steeltoe Logging provider
- Usage
  - Add NuGet references to project
  - Replace Console logging provider with Steeltoe provider
  - Add Logger actuator to service container and pipeline

Loggers Endpoint – Add NuGet References

- Loggers Endpoint
  - Basic functionality - `Steeltoe.Management.Endpoint`
  - ASP.NET Core Actuator - `Steeltoe.Management.EndpointCore`
  - All ASP.NET Core Actuators - `Steeltoe.Management.CloudFoundryCore`

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
    <PackageReference Include="Steeltoe.Management.CloudFoundryCore" Version= "x.y.z" />
<ItemGroup>

</Project>
```

Loggers Endpoint – Use Steeltoe Logging Provider

- Add while building `WebHost`
  - In `ConfigureLogging()`
  - Use `AddDynamicConsole()` on provided `LoggingBuilder`
- `DynamicConsole` is a simple wrapper around standard Console provider
  - Configure using Logging:Console

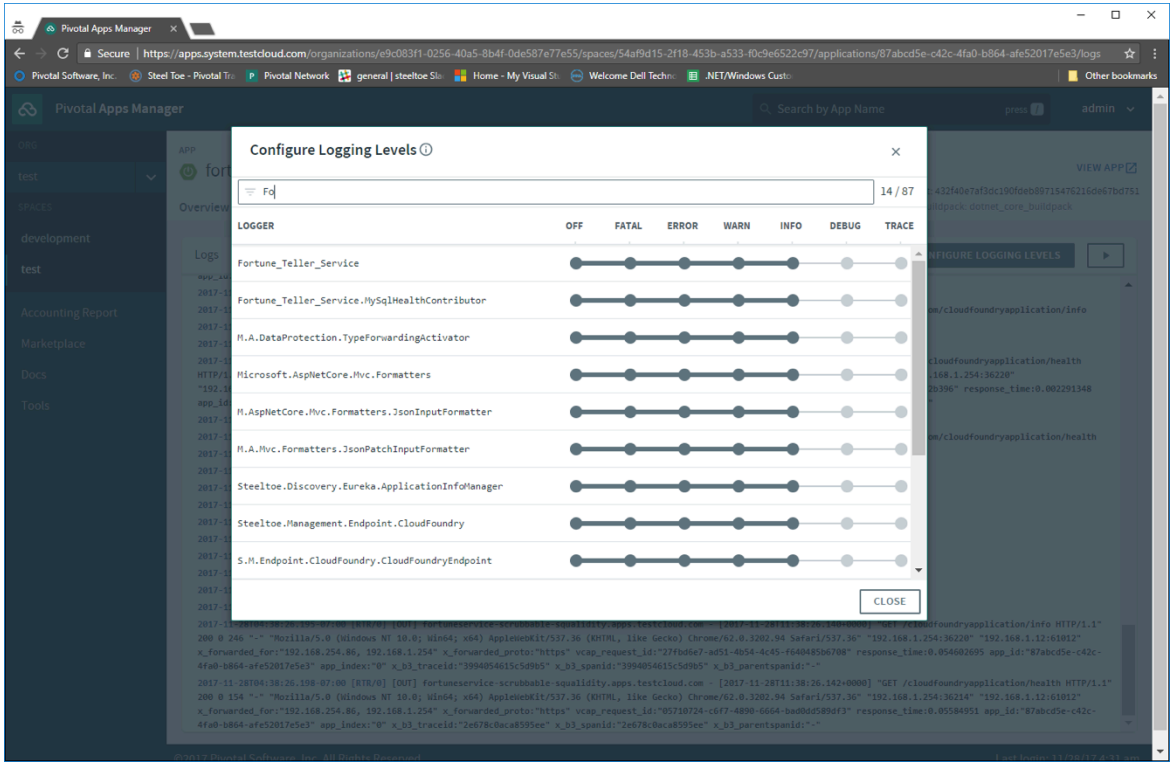
```
public static IWebHost BuildWebHost(string[] args)
{
  var builder = new WebHostBuilder()
    .UseKestrel()
    .UseIISIntegration()
    .ConfigureLogging((hostingContext, logging) =>
    {
      logging.AddConfiguration(
           hostingContext.Configuration.GetSection("Logging"));
      logging.AddDebug();

      logging.AddDynamicConsole(hostingContext.Configuration);

    })
 . . . .
}

public class Startup
{
  public IConfiguration Configuration { get; }
  public void ConfigureServices(IServiceCollection services)
  {
     services.AddOptions();
     services.ConfigureCloudFoundryOptions(Configuration)
  }
}
```

Loggers Endpoint – Add Logger Actuator

- Add Logger endpoint actuator or all actuators
  - `AddLoggerActuator`
  - `AddCloudFoundryActuators`
- Add Logger endpoint middleware to request pipeline
  - `UseLoggerActuator`
  - `UseCloudFoundryActuators`

```
public class Startup
{
  public IConfiguration Configuration { get; }

  public void ConfigureServices(IServiceCollection services)
  {
    services.AddOptions();

    // services.AddLoggerActuator(Configuration);
    services.AddCloudFoundryActuators(Configuration);
}

  public void Configure(IApplicationBuilder app, IHostingEnvironment env)
  {
    .....

    // app.UseLoggerActuator();
    app.UseCloudFoundryActuators();

    app.UseMvc();

}

}
```

Loggers Endpoint - Using on Cloud Foundry



Steeltoe Trace Endpoint

- Used to view the last several requests and responses handled by an application
- Usage
  - Add NuGet references to project
  - Add Trace actuator to service container and pipeline

Trace Endpoint – Add NuGet References

- Trace Endpoint
  - Basic functionality - `Steeltoe.Management.Endpoint`
  - ASP.NET Core Actuator - `Steeltoe.Management.EndpointCore`
  - All ASP.NET Core Actuators - `Steeltoe.Management.CloudFoundryCore`

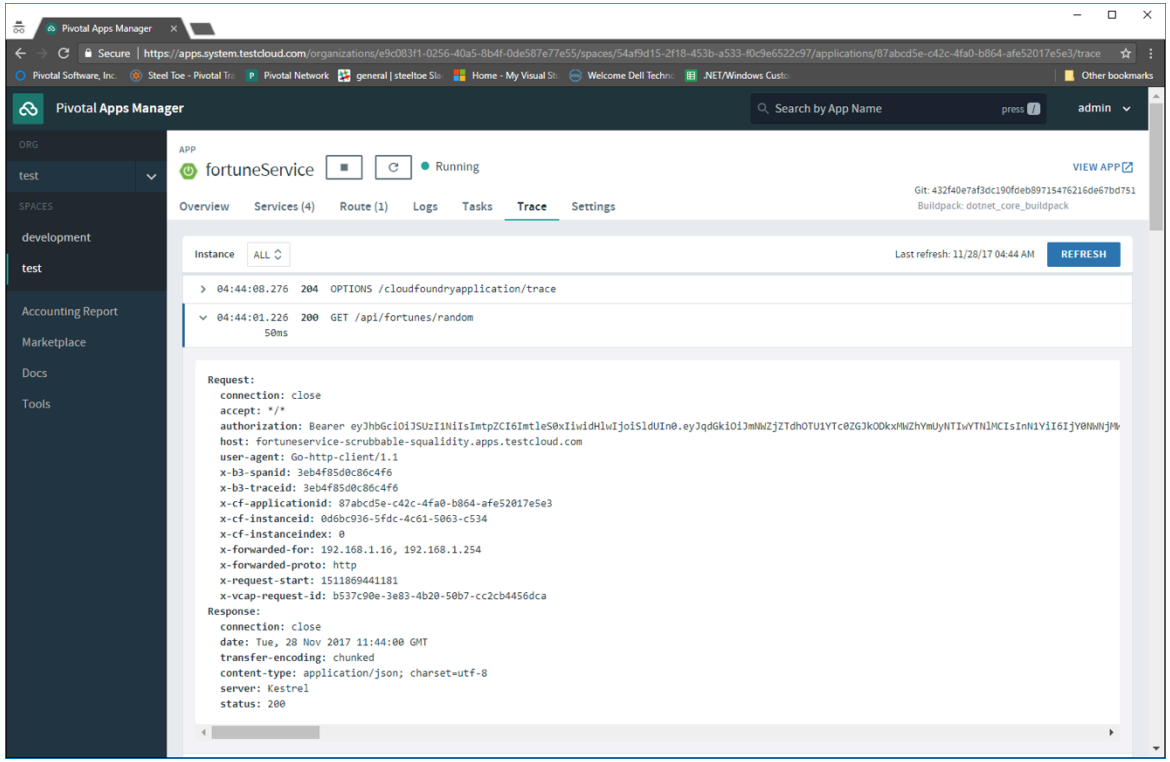
```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
    <PackageReference Include="Steeltoe.Management.CloudFoundryCore" Version= "x.y.z" />
<ItemGroup>

</Project>
```

Trace Endpoint – Add Trace Actuator

- Add Trace endpoint actuator or all actuators
  - `AddTraceActuator`
  - `AddCloudFoundryActuators`
- Add Trace endpoint middleware to request pipeline
  - `UseTraceActuator`
  - `UseCloudFoundryActuators`

```
public class Startup
{
  public IConfiguration Configuration { get; }

  public void ConfigureServices(IServiceCollection services)
  {
    services.AddOptions();

    // services.AddTraceActuator(Configuration);
    services.AddCloudFoundryActuators(Configuration);
}

  public void Configure(IApplicationBuilder app, IHostingEnvironment env)
  {
    .....

    // app.UseTraceActuator();
    app.UseCloudFoundryActuators();

    app.UseMvc();

}

}
```

Trace Endpoint - Using on Cloud Foundry



Steeltoe Cloud Foundry Endpoint

- Used to enable integration of the management endpoints with the Pivotal Apps Manager
  - Exposes query endpoint which returns the Ids and links to all enabled management endpoints
  - Adds security middleware to the request pipeline which requires using security tokens acquired from the UAA to access endpoints
- Usage
  - Add NuGet references to project
  - Configure settings
  - Add CloudFoundry actuator to service container and pipeline

Cloud Foundry Endpoint – Add NuGet References

- Trace Endpoint
  - Basic functionality - `Steeltoe.Management.Endpoint`
  - ASP.NET Core Actuator - `Steeltoe.Management.EndpointCore`
  - All ASP.NET Core Actuators - `Steeltoe.Management.CloudFoundryCore`

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
    <PackageReference Include="Steeltoe.Management.CloudFoundryCore" Version= "x.y.z" />
<ItemGroup>

</Project>
```

Cloud Foundry Endpoint – Endpoint Settings

- For integration with Pivotal Apps Manager, you must configure the management query endpoint to `/cloudfoundryapplication`
  - Use `management:endpoints:path`
- You may need to disable certificate validation depending on your installation of Cloud Foundry
  - `management:endpoints:cloudfoundry:validateCertificates` – set to false to disable

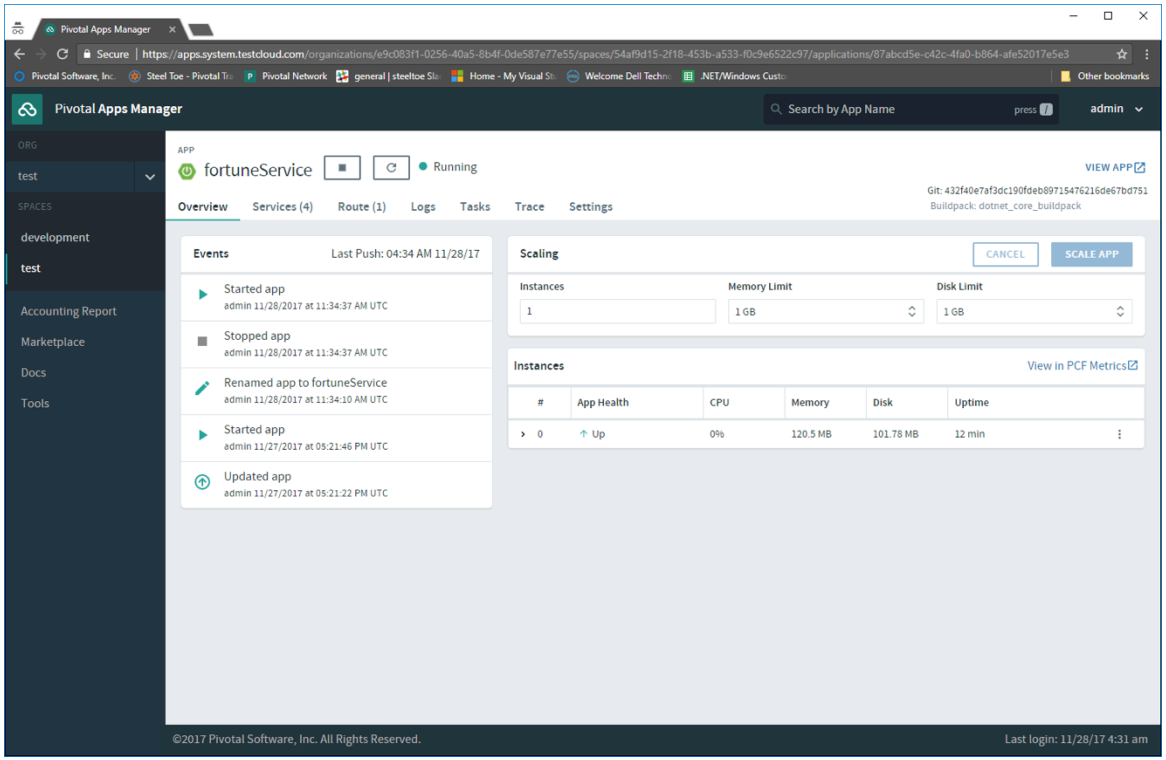
```
management:
  endpoints:
    path: /cloudfoundryapplication
    cloudfoundry:
      validateCertificates: false
```

CloudFoundry Endpoint – Add Actuators

- Add CloudFoundry endpoint actuator or all actuators
  - `AddCloudFoundryActuator`
  - `AddCloudFoundryActuators`
- Add CloudFoundry endpoint middleware to request pipeline
  - `UseCloudFoundryActuator`
  - `UseCloudFoundryActuators`

```
public class Startup
{
  public IConfiguration Configuration { get; }

  public void ConfigureServices(IServiceCollection services)
  {
    services.AddOptions();

    // services.AddTraceActuator(Configuration);
    // services.AddLoggerActuator(Configuration);
    // services.AddCloudFoundryActuator(Configuration);

    services.AddCloudFoundryActuators(Configuration);
}

  public void Configure(IApplicationBuilder app, IHostingEnvironment env)
  {
    .....
    // app.UseCloudFoundryActuator();
    // app.UseTraceActuator();
    // app.UseLoggerActuator();

    app.UseCloudFoundryActuators();

    app.UseMvc();

}

}
```

Cloud Foundry Endpoint - Using on Cloud Foundry



Fortune Teller Demo – Bringing it all together!