Steeltoe

Steeltoe Project

- Open Source
  - https://github.com/SteeltoeOSS
  - Code repos are functionally organized (Configuration, Discovery, Connectors, etc.)
- .NET support (NetStandard 2.0)
  - .NET Core (Windows, Linux & OSX)
  - .NET Framework
- Application type support
  - ASP.NET 4
  - ASP.NET Core
  - Console apps

Steeltoe Project

Steeltoe
http://steeltoe.io

Enable building Cloud Native Applications using .NET

- Simplify implementing common distributed system patterns
  - Configuration providers (e.g. Config Server, etc.)
  - Service Discovery (e.g. Eureka, Consul, etc.)
  - Client-side Load Balancing
  - Fault Tolerance(e.g. Netflix Hystrix)
  - Distributed Tracing (e.g. OpenCensus)
  - Observability (e.g. Actuators, OpenCensus)
- Simplify using .NET & ASP.NET on Cloud platforms
  - 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)
- Provide tools to aid in developer productivity
  - Develop and test locally
    - Docker images for common backing services
    - Push to cloud platform …. No code or config changes

Enable .NET developers to easily leverage OSS cloud infrastructure

Steeltoe Project

- 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.ZBase – base X.Y.Z functionality, application type independent
    - e.g. `Steeltoe.Extensions.Configuration.CloudFoundryBase`
  - Steeltoe.X.Y.ZCore – .NET Core DI support (e.g. ASP.NET Core)
    - e.g. `Steeltoe.Extensions.Configuration.CloudFoundryCore`
  - Steeltoe.X.Y.ZAutofac – ASP.NET 4 Autofac DI support
    - e.g. `Steeltoe.Extensions.Configuration.CloudFoundryAutofac`

Steeltoe Project

- Documentation – http://steeltoe.io
- Samples – https://github.com/SteeltoeOSS/Samples
  - Single function - organized by framework area (e.g. Configuration, Discovery, etc.)
    - ASP.NET 4, ASP.NET Core & Console based 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
    - WorkshopFinal – Fortune Teller using all of the Steeltoe components
  - eShopContainers - https://github.com/SteeltoeOSS/eShopOnContainers
- Slack - https://slack.steeltoe.io/

Steeltoe Sample

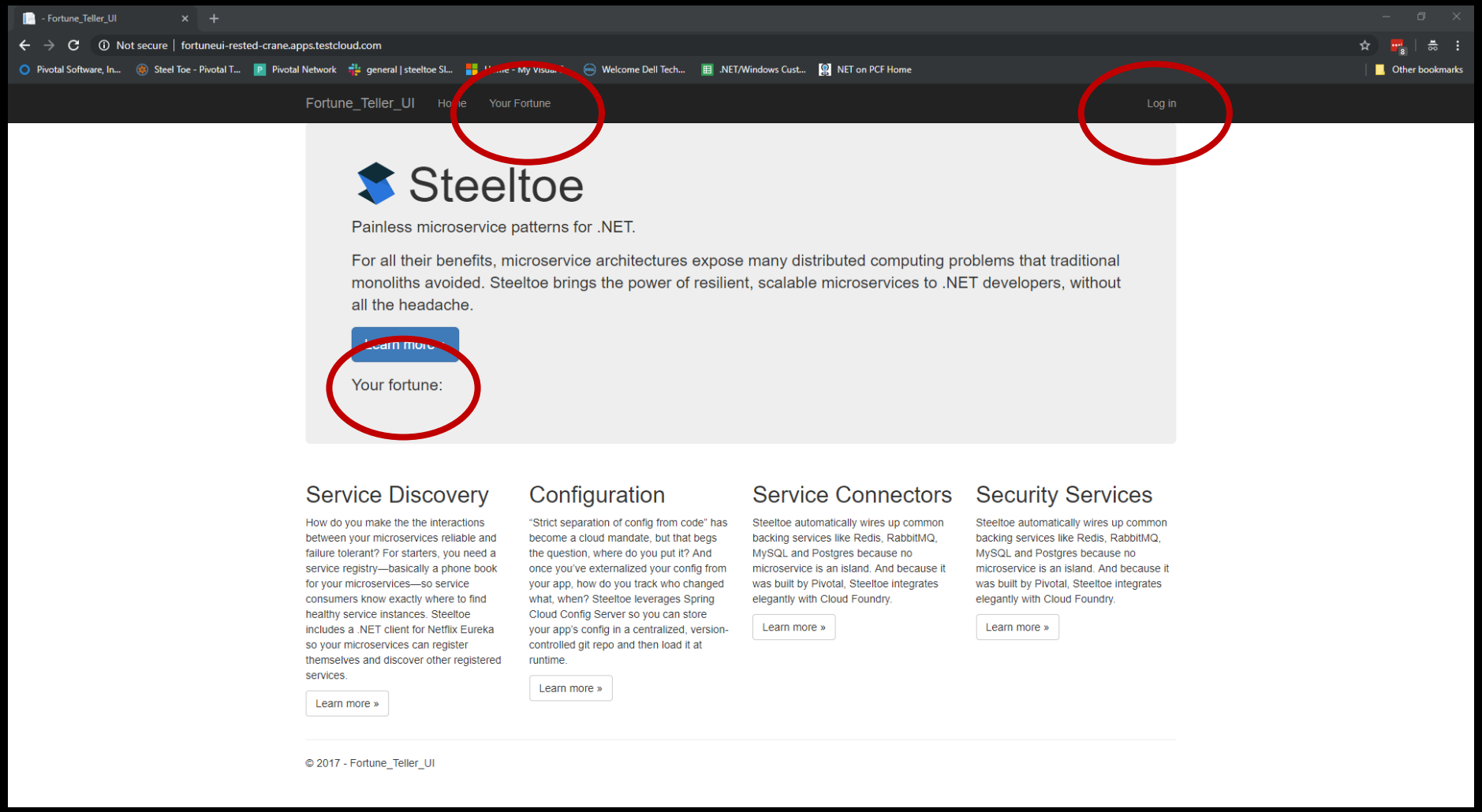
Fortune Teller

Steeltoe
http://steeltoe.io

https://github.com/SteeltoeOSS/Samples/tree/dev/WorkshopFinal

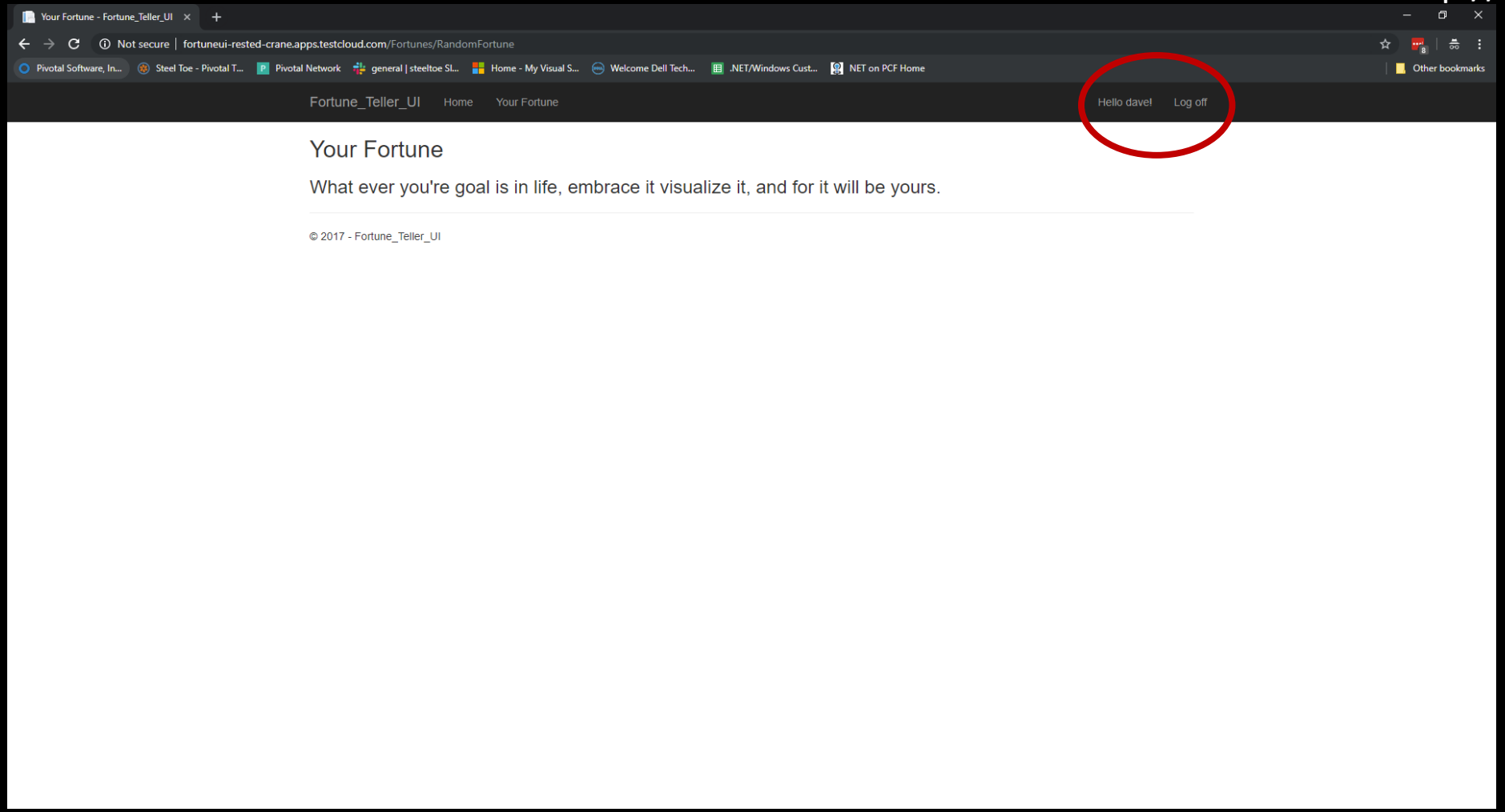
Fortune Teller

Steeltoe
http://steeltoe.io



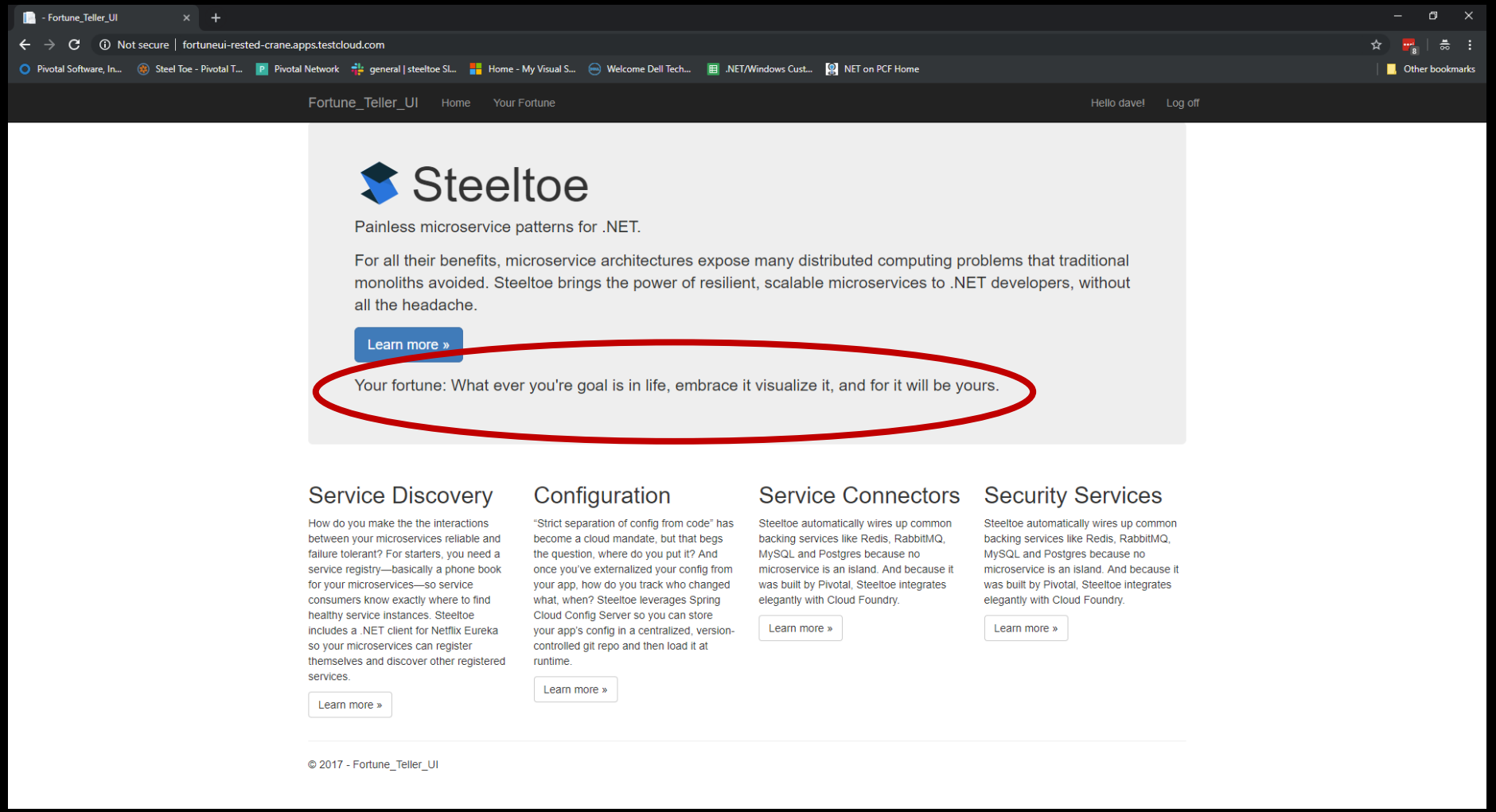
Fortune Teller

Steeltoe
http://steeltoe.io



Fortune Teller

Steeltoe
http://steeltoe.io



Fortune Teller Design Requirements

Steeltoe
http://steeltoe.io

- Infrastructure independent
  - Cloud platform (e.g. vSphere, Azure, AWS, GCP, etc.)
  - Operating system (e.g. Windows, Linux, etc.)
  - Polyglot (e.g. .NET, Java, Node, etc.)
- Leverage the cloud platform services
  - Database, Caching, Security, Service Registry, Config, etc.
- Use modern microservices patterns
  - Externalized configuration
  - Platform Service abstractions
  - Service Discovery & Load Balancing
- Scale Horizontally
- Reliable - Fault tolerant
  - Command pattern
  - Bulkhead pattern
  - Circuit Breaker pattern
- Secure
  - RBAC
  - Single-Sign-on
- Observable – Monitor in production
  - Health, logs, metrics, traces, circuits, fallbacks, etc.

Infrastructure Independent Bosh & Cloud Foundry PAS

Steeltoe
http://steeltoe.io

Users

Dev

Dev

IT Ops

IT Ops

IT Ops

Cloud Native Platform

App Frameworks

Contract: **12 Factor App**

App Platform

Contract: **BOSH Release**

Infrastructure Automation

Contract: **Cloud Provider Interface**

Infrastructure

Components

Spring Boot

Spring Cloud

Spring Cloud Dataflow

Steeltoe

.NET Core

.NET

Cloud Foundry PAS

SSO Service

Discovery Service

Configuration Service

Metrics & Monitoring Service

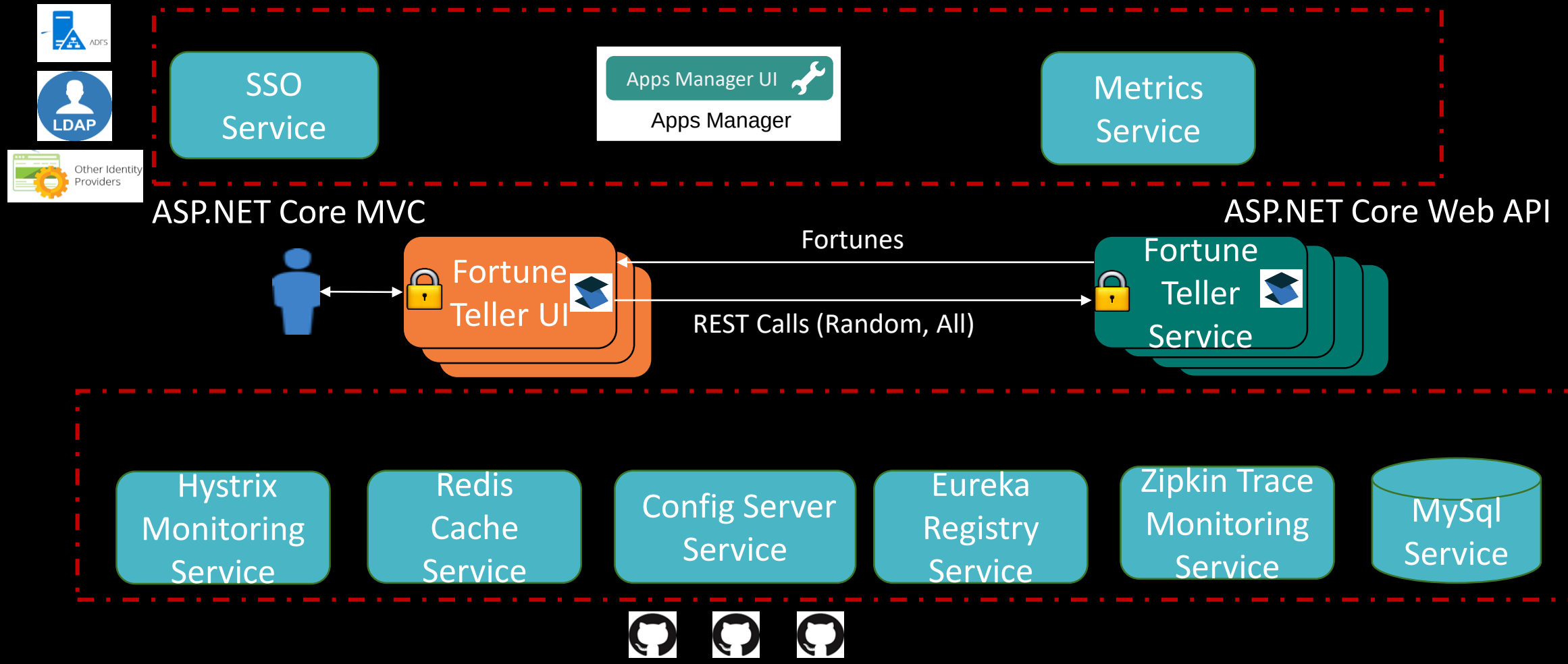
Cache Service

Database Services

BOSH

Leverage Cloud Platform Services
Cloud Foundry Marketplace

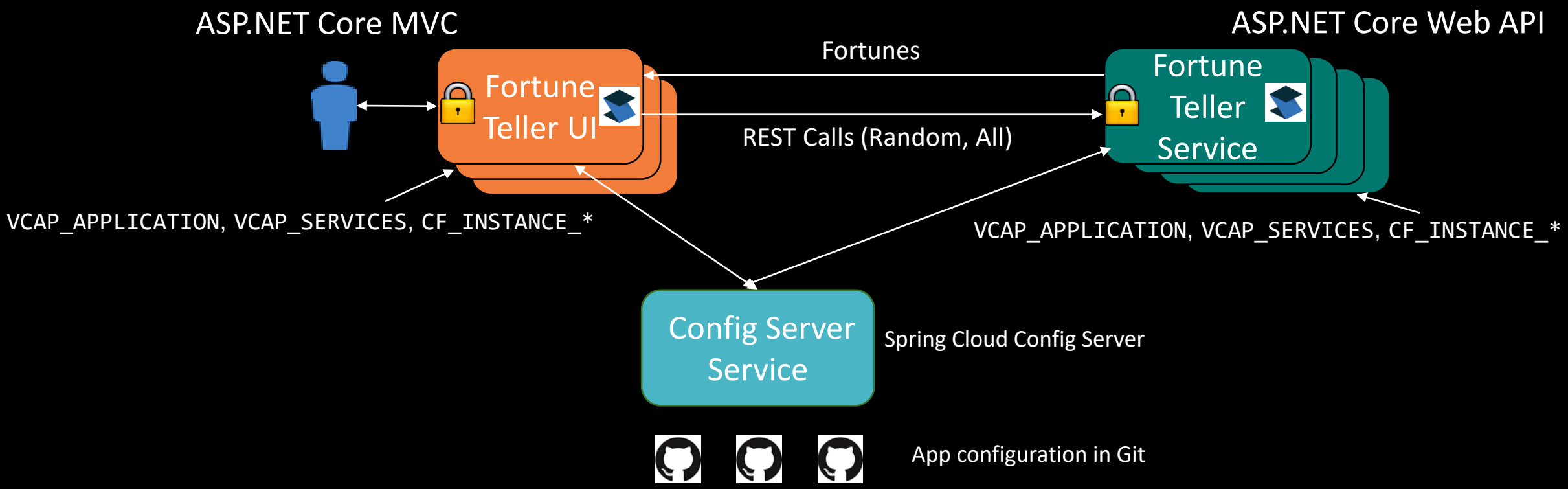
Steeltoe
http://steeltoe.io



Externalized Configuration Pattern

Steeltoe
http://steeltoe.io

"Enable services to run in multiple environments without modification"



Externalized Configuration
Cloud Foundry Environment Variables

- Used to communicate application environment & configuration to deployed containers
  - VCAP_APPLICATION
    - Application attributes – version, instance index, limits, URLs, etc.
  - VCAP_SERVICES
    - Bound services – name, label, credentials, etc.
  - CF_INSTANCE_*
    - CF_INSTANCE_ADDR, CF_INSTANCE_INDEX, etc.

Externalized Configuration
VCAP_APPLICATION Environment Variable


```
"VCAP_APPLICATION": {
    "cf_api": "https://api.system.testcloud.com",
    "limits": {
     "fds": 16384
    },
    "application_name": "fortuneService",
    "application_uris": [
     "fortuneservice-lean-tasmaniandevil.apps.testcloud.com"
    ],
    "name": "fortuneService",
    "space_name": "test",
    "space_id": "cc682e26-2155-4649-b948-4f0d02f887da",
    "uris": [
     "fortuneservice-lean-tasmaniandevil.apps.testcloud.com"
    ],
    "users": null,
    "application_id": "d32540f4-a59a-47d7-97fe-c998fb20baad“
}
```

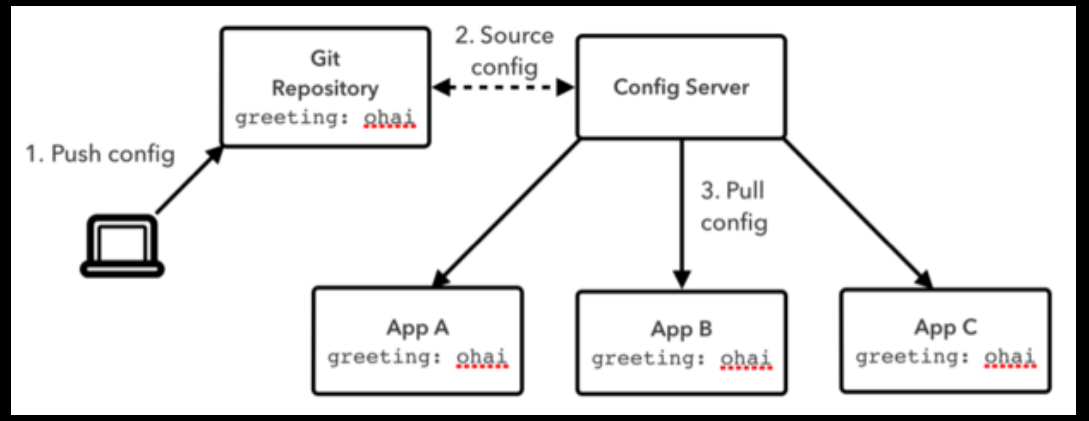
Externalized Configuration
VCAP_SERVICES Environment Variable


```
"VCAP_SERVICES": {
  "p.mysql": [{
    "name": "myMySqlService",
    "instance_name": "myMySqlService",
    "binding_name": null,
    "credentials": {
     "hostname": "q-n2s3y1.q-g74.bosh",
     "jdbcUrl": "jdbc:mysql://q-n2s3y1.q-g74.bosh:3306/service_instance_db?user=8dd22f298fab4cd6895be59605dcb49a&password=lpuicoa1dfr2xj4b&useSSL=false",
     "name": "service_instance_db",
     "password": "lpuicoa1dfr2xj4b",
     "port": 3306,
     "uri": "mysql://8dd22f298fab4cd6895be59605dcb49a:lpuicoa1dfr2xj4b@q-n2s3y1.q-g74.bosh:3306/service_instance_db?reconnect=true",
     "username": "8dd22f298fab4cd6895be59605dcb49a"
    },
    "syslog_drain_url": null,
    "volume_mounts": [],
    "label": "p.mysql",
    "provider": null,
    "plan": "db-small",
    "tags": [
     "mysql"
    ]
   }],
```

Externalized Configuration Spring Cloud Config Server

Steeltoe
http://steeltoe.io

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



Externalized Configuration
Steeltoe Configuration Providers

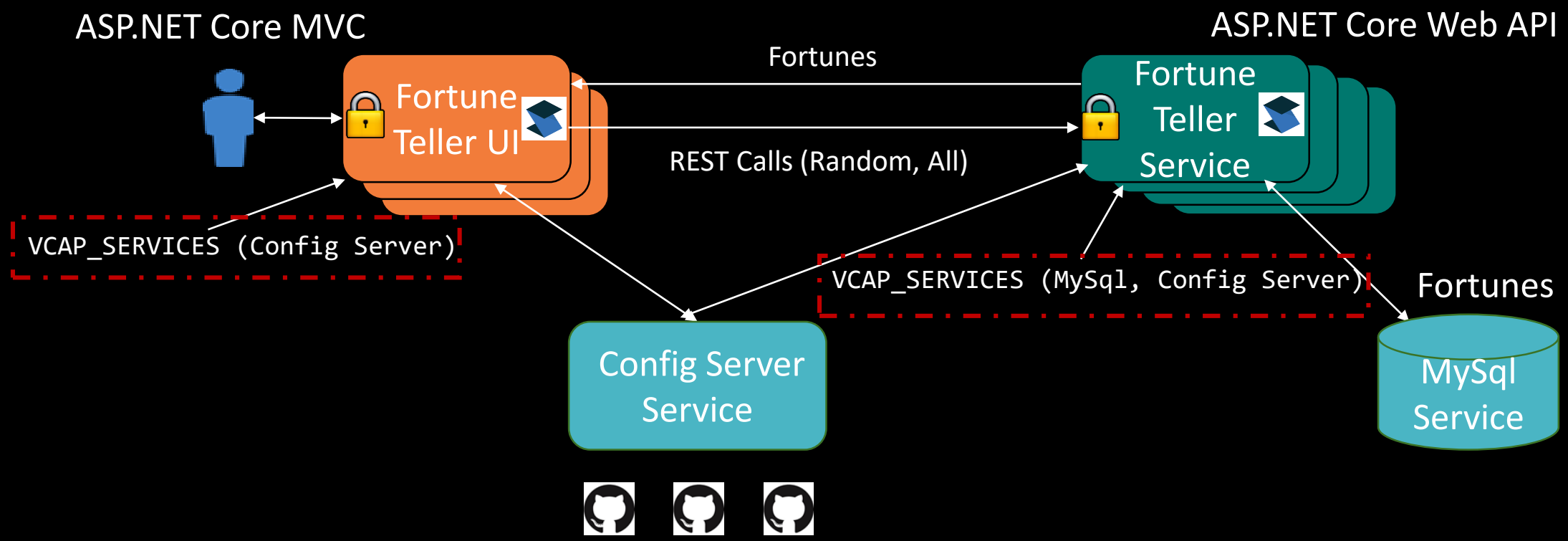
- Simplify adding additional sources of application configuration data into .NET applications
  - Built on .NET Configuration and Options packages
- Several Configuration Providers
  - CloudFoundry – adds configuration values from VCAP_*, CF_* environment variables
  - Config Server Client – adds configuration values from Config Server Service
  - Placeholder Resolver – use and resolve configuration placeholders
    - key = ${some:other:key?default_value}
  - RandomValue Generator – generate random values
    - key=${some:other:key?${random.string}}
- Usage
  - Add provider to Configuration Builder (e.g. `AddConfigServer()`)
  - Configure any needed settings
  - Optionally, configure any Options classes (i.e. `Configure<YourOptionsClass>()`)
  - Access config values using Options or indexers
  - For Config Server on Cloud Foundry, create & bind service instance for usage

To the Sample

Platform Service Abstraction Pattern

Steeltoe
http://steeltoe.io

"Avoid tight coupling to platform provided services"



Platform Service Abstractions
VCAP_SERVICES Environment Variable


```
"VCAP_SERVICES": {
  "p.mysql": [{
    "name": "myMySqlService",
    "instance_name": "myMySqlService",
    "binding_name": null,
    "credentials": {
     "hostname": "q-n2s3y1.q-g74.bosh",
     "jdbcUrl": "jdbc:mysql://q-n2s3y1.q-g74.bosh:3306/service_instance_db?user=8dd22f298fab4cd6895be59605dcb49a&password=lpuicoa1dfr2xj4b&useSSL=false",
     "name": "service_instance_db",
     "password": "lpuicoa1dfr2xj4b",
     "port": 3306,
     "uri": "mysql://8dd22f298fab4cd6895be59605dcb49a:lpuicoa1dfr2xj4b@q-n2s3y1.q-g74.bosh:3306/service_instance_db?reconnect=true",
     "username": "8dd22f298fab4cd6895be59605dcb49a"
    },
    "syslog_drain_url": null,
    "volume_mounts": [],
    "label": "p.mysql",
    "provider": null,
    "plan": "db-small",
    "tags": [
     "mysql"
    ]
   }],
```

Platform Service Abstractions
Steeltoe Connectors

- Abstracts away and simplifies using Cloud Foundry provided services
  - When application pushed to Cloud Foundry, service bindings are auto detected and used
  - Can also configure settings using `appsettings.json` for local development; settings overridden when app is pushed to Cloud Foundry
  - Adds appropriate objects into the service container (e.g. `Connections`, `Connection factories`, or `DbContext`)
- Several Connectors
  - MySQL
  - Redis
  - Postgres
  - RabbitMQ
  - SqlServer
  - MongoDB

Service Abstractions
Steeltoe MySql Connector

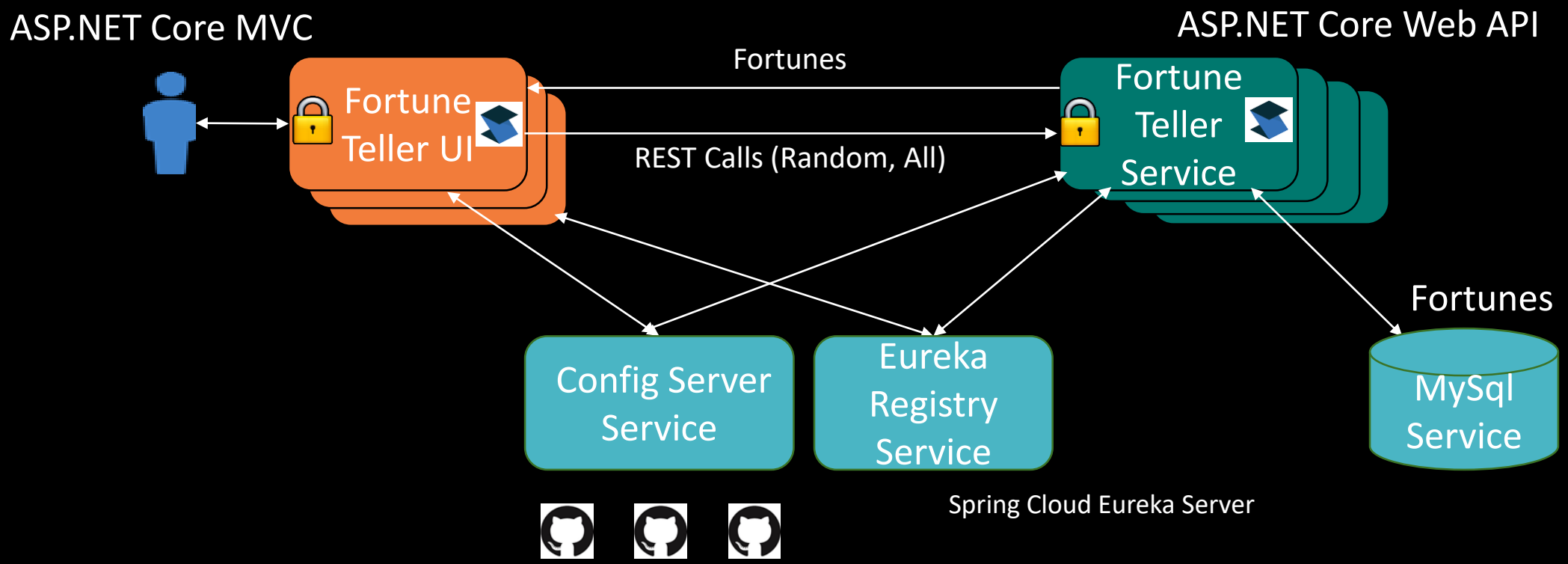
- Configures and adds `MySqlConnection/IDbConnection` or `DbContext` to the container
  - Supports Oracle and Open Source ADO.NET providers
  - Supports both EntityFramework and EntityFrameworkCore `DbContext`
- Usage
  - Configure any needed MySql connector settings in appsettings.json, Config Server, etc.
  - Add `MySqlConnection` or `DbContext` to service container
  - Inject `MySqlConnection` or `DbContext` into controller/view and use
  - On Cloud Foundry, create & bind MySql service instance for access

To the Sample

Service Discovery Pattern

Steeltoe
http://steeltoe.io

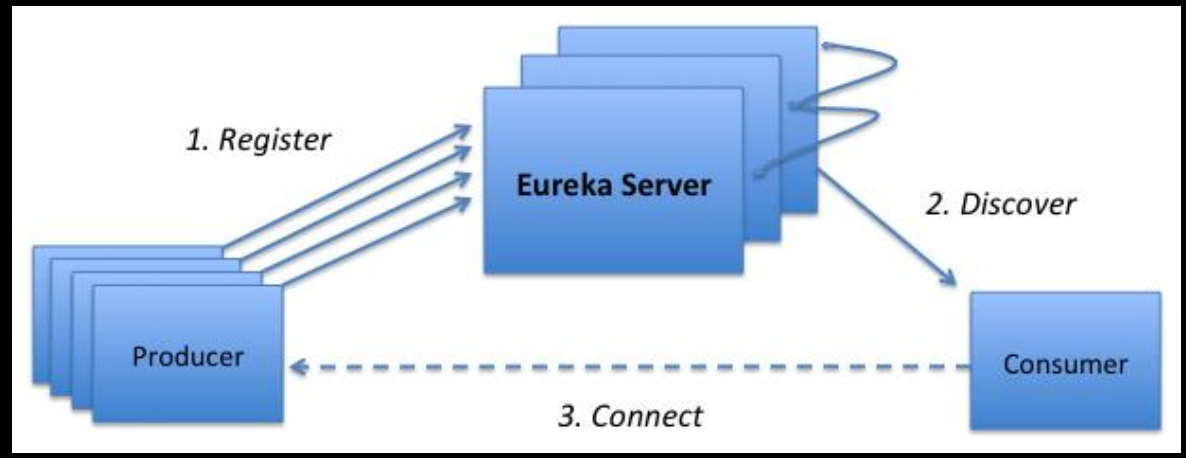
"Enable clients to make requests of a dynamically changing set of service instances"



Service Discovery
Spring Cloud Eureka Server

Steeltoe
http://steeltoe.io

- Server for registering and discovering services
  - Exposes a REST based API http://localhost:8761/eureka
- Applications register addresses by service 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

Service Registration
Spring Cloud Eureka Server

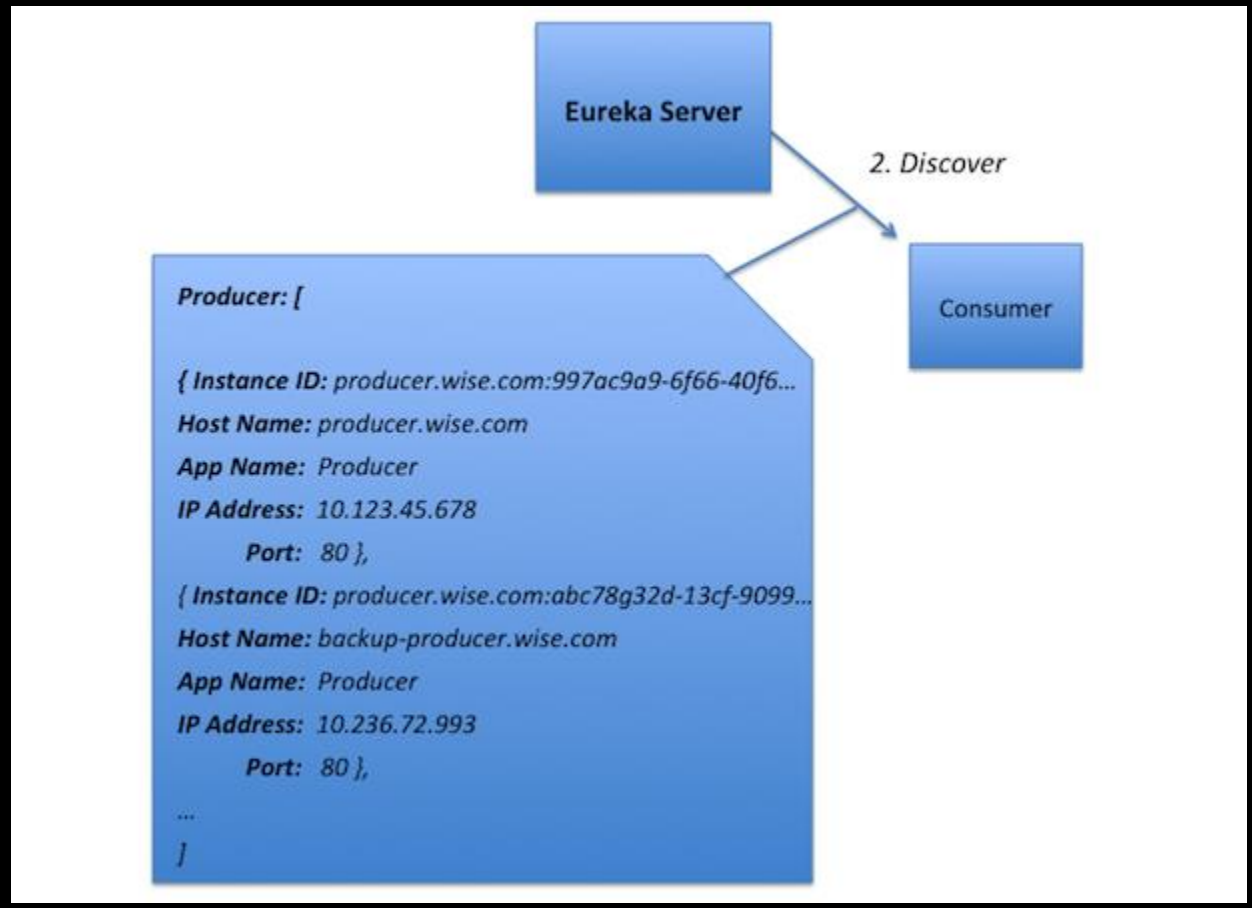
Steeltoe
http://steeltoe.io

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

Service Discovery
Spring Cloud Eureka Server

Steeltoe
http://steeltoe.io

- 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



Service Discovery Steeltoe Discovery providers

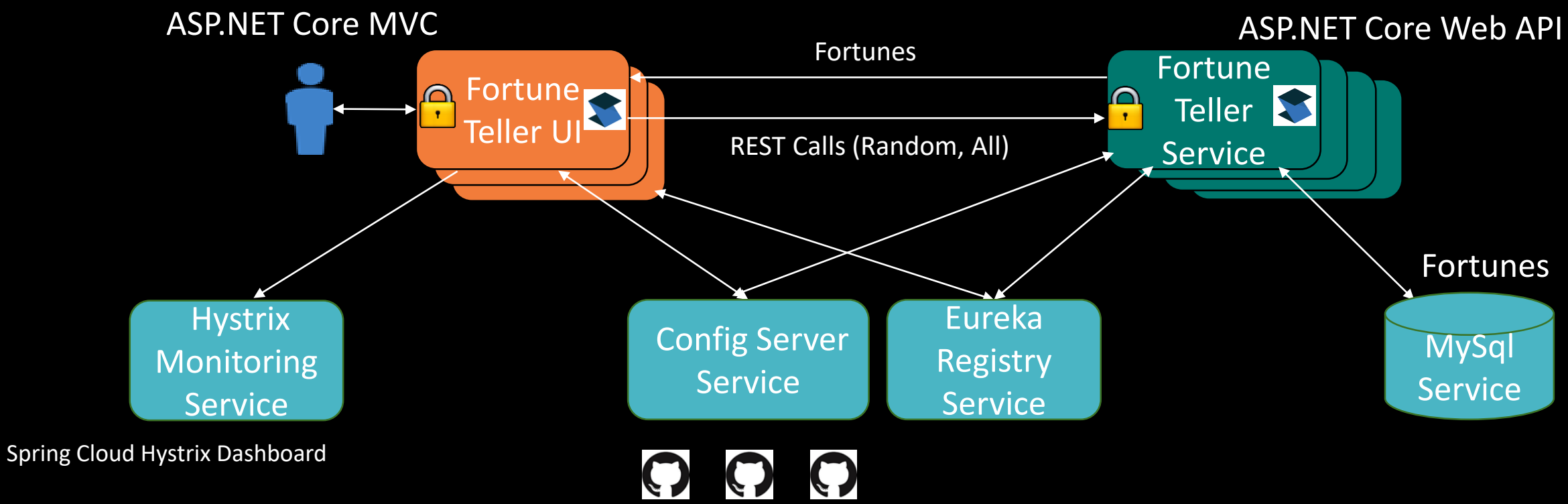
- Provides a configurable generalized interface for Service Registry interaction
  - `Steeltoe.Common.Discovery.IDiscoveryClient`
- Providers
  - Eureka – client for Netflix / Spring Cloud Eureka Server
  - Consul - client for Hashicorp Consul Server
- Usage
  - Add Discovery Client to service container (i.e. `AddDiscoveryClient()`)
  - Configure Discovery Client settings for provider
  - Start the Discovery Client (i.e. `UseDiscoveryClient()`)
  - Inject a `IDiscoveryClient` and use it to lookup services
  - Optionally use Steeltoe HTTPHandler/Load Balancer or HttpClientFactory to do lookups
  - On Cloud Foundry, create & bind Eureka service instance for access

To the Sample

Fault Tolerance

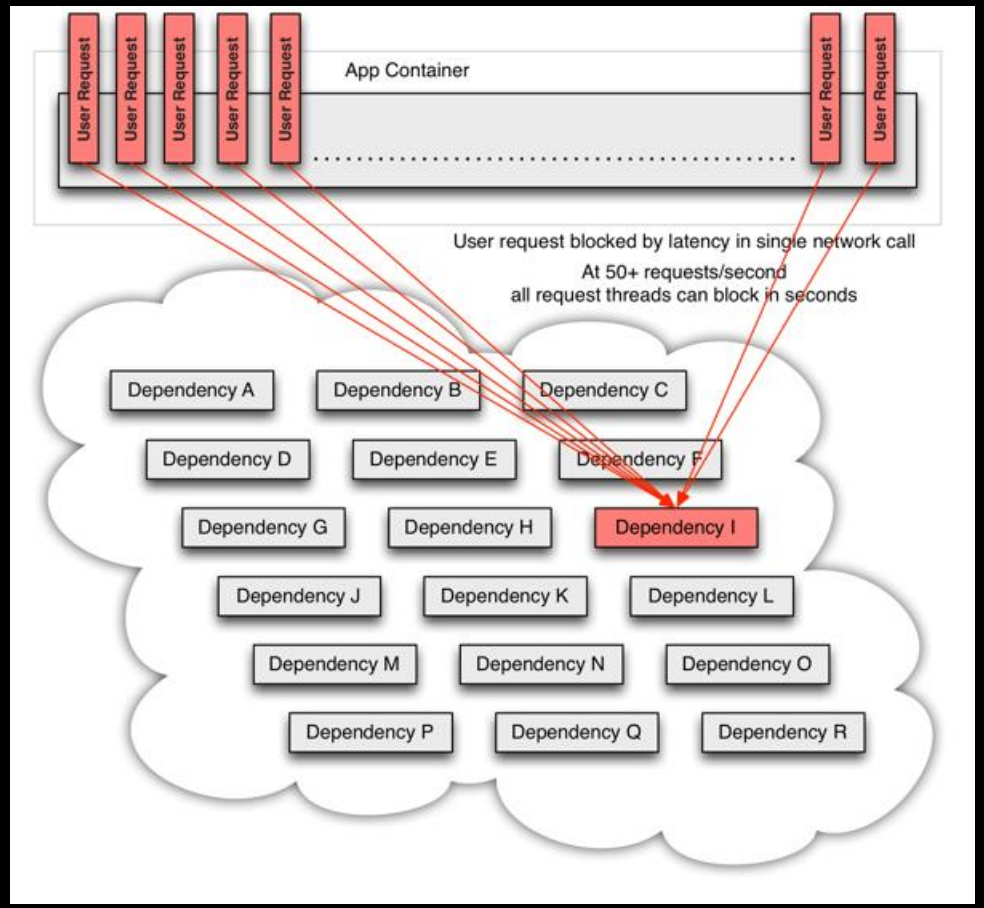
Steeltoe
http://steeltoe.io

"Prevent network or service failures from cascading and impacting other services"

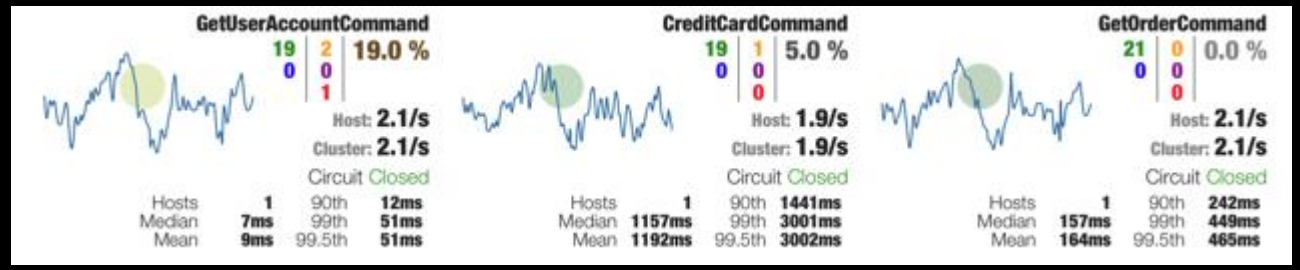


Fault Tolerance – Hystrix Framework

Steeltoe
http://steeltoe.io



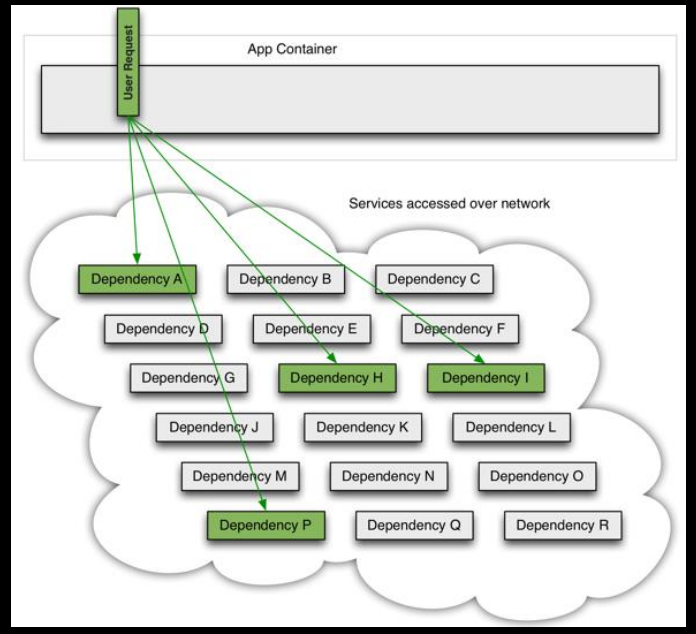
- Hystrix consists of two parts
  - Fault tolerance framework for dealing with failures and latency in distributed systems
    - Utilize isolation patterns to limit impacts
  - Near real-time monitoring & alerting
    - Shorten time-to-discover and time-to-recover from failures
- Good source for understanding Hystrix is the Netflix Wiki
  - https://github.com/Netflix/Hystrix/wiki



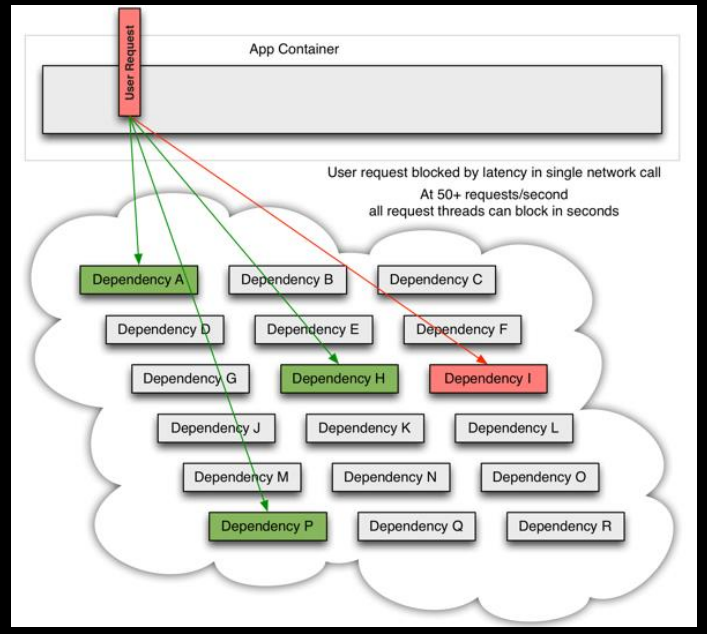
Fault Tolerance – Hystrix Framework

Steeltoe
http://steeltoe.io

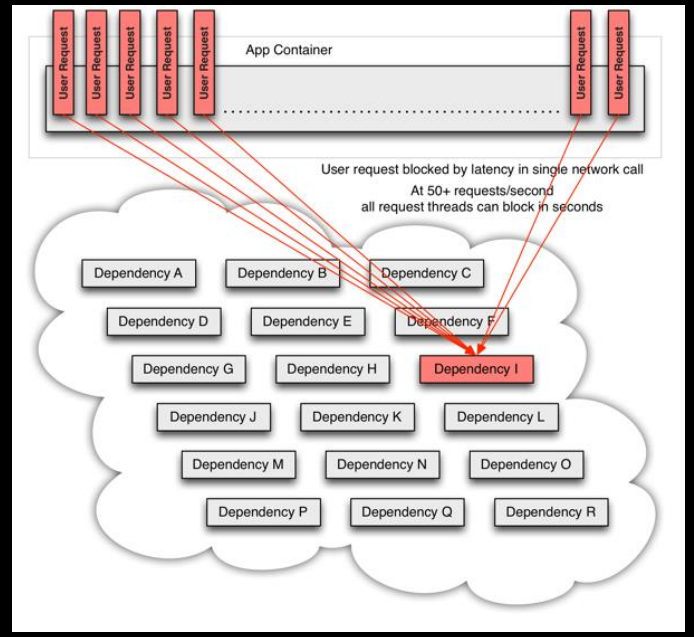
Healthy System



Backend Dependency Becomes Latent

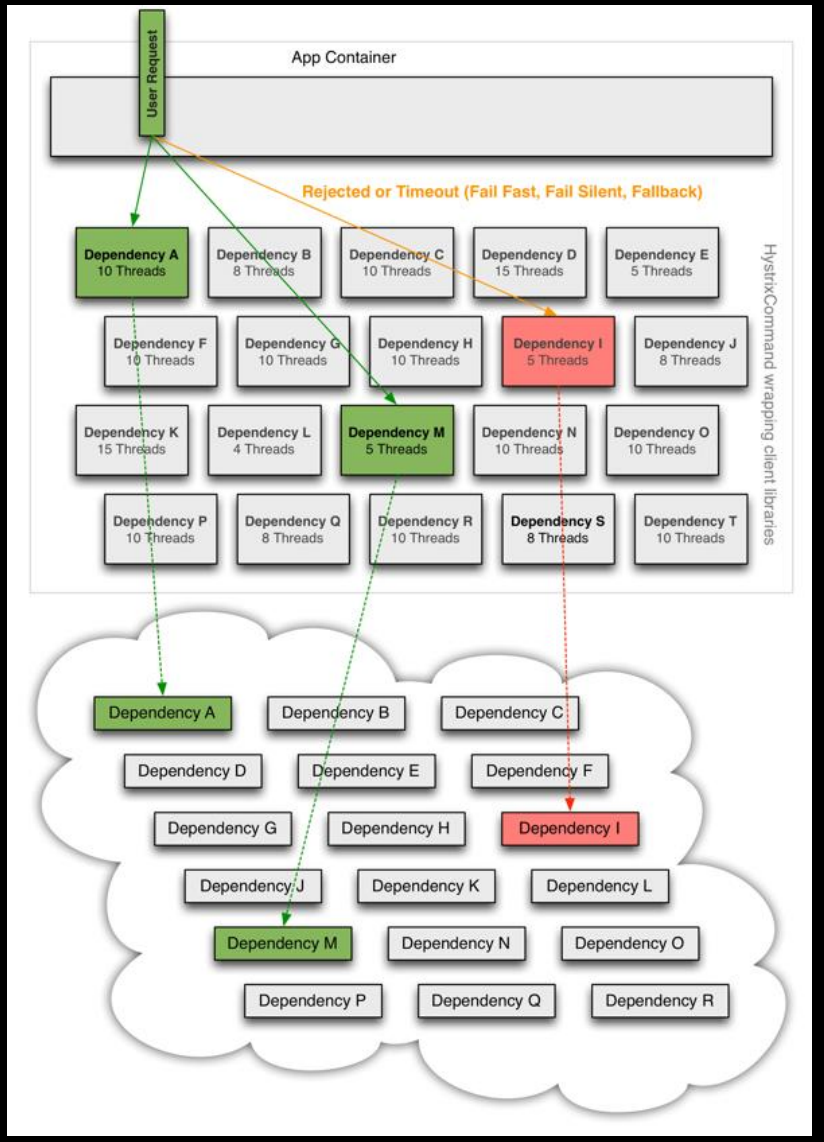


Latency cascades resources become saturated



Fault Tolerance – Hystrix Framework

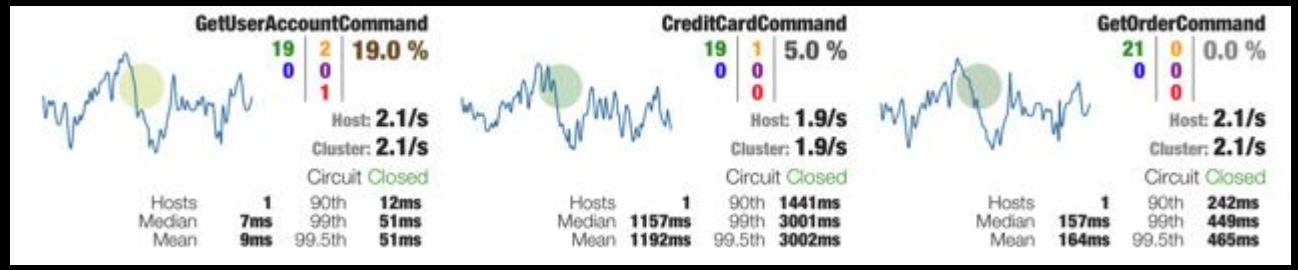
- Fault tolerance framework
  - Wrap all external dependencies in Hystrix commands
    - Requests executed on separate thread
    - Automatically execute fallback logic on failures or timeouts
  - Configurable thread pools for each dependency
    - Fail fast if pool consumed
  - Measures success, failures,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

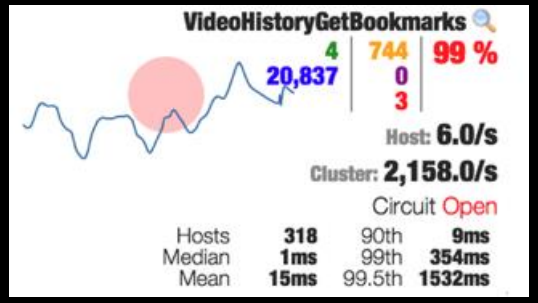


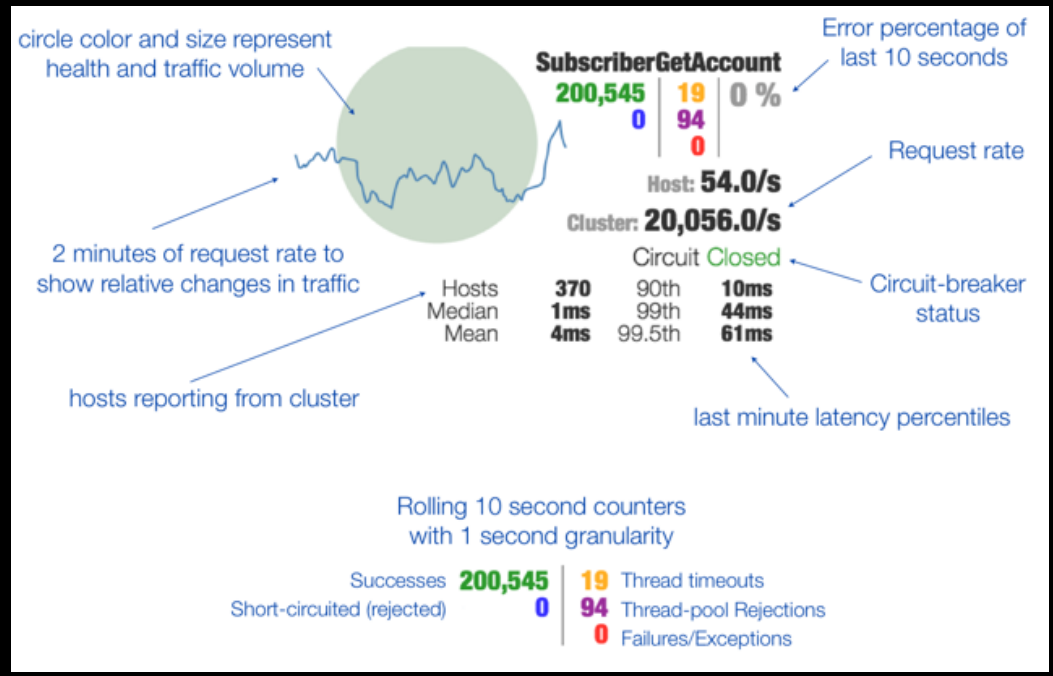
Fault Tolerance
Hystrix Monitoring and Alerting

Steeltoe
http://steeltoe.io

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







Fault Tolerance - Steeltoe Hystrix

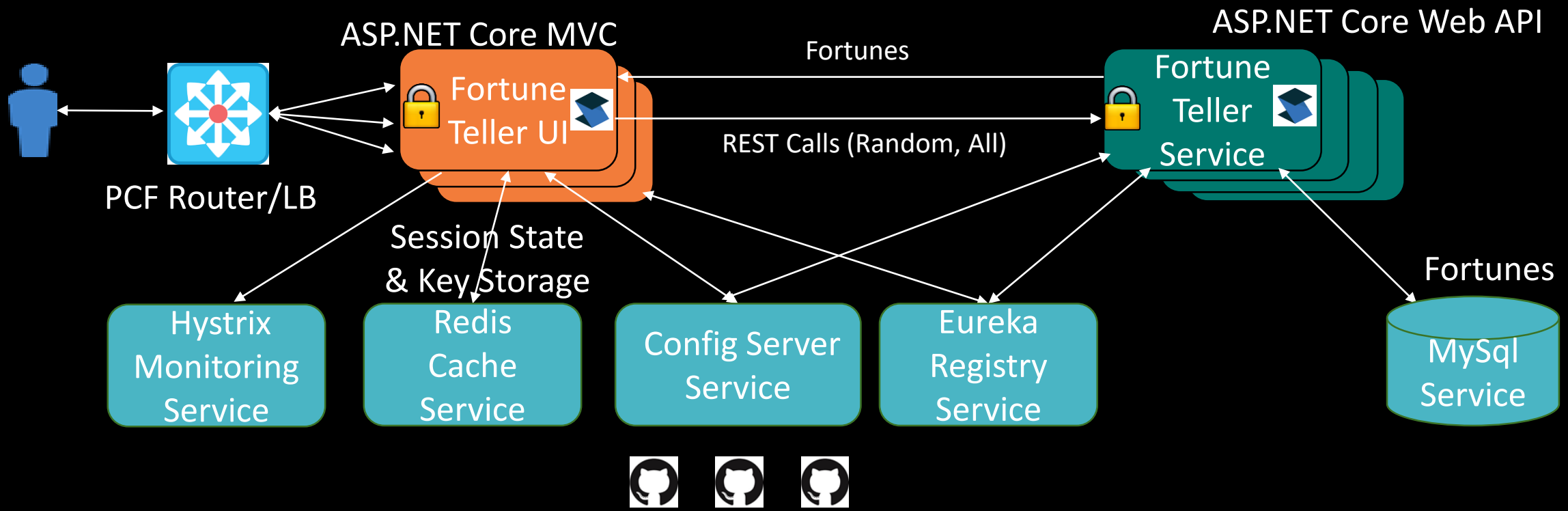
- Provide an implementation of the Hystrix command pattern for .NET applications
  - Built using .NET Tasks & underlying .NET thread pool
  - Includes monitoring (i.e. metrics, status, etc.) compatible with Spring Cloud Services Dashboard
- Usage
  - Define HystrixCommand(s) which wrap external dependencies
    - Code RunAsync() and RunFallbackAsync()
  - Add HystrixCommand(s) to service container (i.e. `AddHystrixCommand()`)
    - Can also new HystrixCommand as needed
  - Optionally, add and use Hystrix Metrics stream
  - Configure any needed settings
  - Use HystrixCommand(s) to access dependent resources
  - On Cloud Foundry, create & bind Hystrix Monitoring service instance for usage

To the Sample

Scaling Horizontally

Steeltoe
http://steeltoe.io

"Ensure all application and user state is stored externally "



Scaling Horizontally
Session State Issue

- We are using ASP.NET Core session middleware for managing session state (i.e. Users current fortune)
  - Session state is stored in dictionary
  - Session ID used to save and fetch state
  - Session IDs are stored in cookie & sent to browser
  - Session IDs are encrypted using DataProtection services before adding to cookie
  - Dictionary saved in `IDistributedCache` - defaults to In-Memory cache
    - Issue: When scaling the UI horizontally, users would lose session state

Scaling Horizontally
DataProtection Key Store Issue

- DataProtection is a crypto service used for protecting data
  - Used both internally by ASP.NET Core and optionally by application code
  - Added as a service using `AddDataProtection()`
  - Automatically generates keys and manages them in a key-ring which it stores in a repository
    - Issue: Default is to store key ring in local file system
  - Fortunately there are extension methods you can use to configure its behavior
    - `PersistKeysToFileSystem(..)`
    - `PersistKeysToRedis(…)`

Scaling Horizontally
Steeltoe Redis Connector

- Can use the Connector to configure both a ASP.NET Core `RedisCache` and a StackExchange `IConnectionMultiplexer`
  - `Adds RedisCache and IConnectionMultiplexer` to container
  - `RedisCache` implements `IDistributedCache`
    - Session service will use it to store users fortune
- Usage
  - Configure any Redis client settings as needed
  - Add DistributedRedisCache to service container
  - Optionally, inject `IDistributedCache` and/or `IConnectionMultiplexer` and use for other purposes
  - On Cloud Foundry, create & bind Redis service instance for usage

Scaling Horizontally
Steeltoe Redis Key Storage Provider

- Configures `DataProtection` to use a Cloud Foundry Redis service for Key Ring storage
  - Requires StackExchange `IConnectionMultiplexer` to be available in the container
- Usage
  - Configure Redis client settings as needed
  - Add `IConnectionMultiplexer` to service container
  - Configure DataProtection to use Redis for Key Storage
  - On Cloud Foundry create & bind Redis service instance for usage

To the Sample

Security

Steeltoe
http://steeltoe.io

"Enable SSO and secure REST endpoints"

SSO Service

OAuth2

ASP.NET Core MVC

ASP.NET Core Web API

Fortunes

Fortune Teller UI

REST Calls (Random, All)

Fortune Teller Service

Session State & Key Storage

Hystrix Monitoring Service

Redis Cache Service

Config Server Service

Eureka Registry Service

Fortunes

MySql Service

Security - Steeltoe Provider Overview

http://steeltoe.io

- Security providers that simplify using Cloud Foundry based security services
  - Enable using UAA Server and/or Pivotal Single Sign-on (SSO) service for Authentication and Authorization services
- Security Providers
  - OAuth2
  - JSON Web Tokens (JWT)

Security - Steeltoe OAuth2 Provider

- Enables applications to implement an OAuth 2.0 Authorization Code grant flows using security services provided by Cloud Foundry
  - UAA Server
  - Pivotal Single Sign-on service
- Usage
  - Add Authentication and Authorization services to container
  - Add Authentication middleware to pipeline
  - Apply Authorization attributes to controllers to protect actions
  - Implement token forwarding as needed
  - Configure any OAuth2 provider settings you need
  - On Cloud Foundry create & bind OAuth service instance for usage

Security - Steeltoe JWT Provider

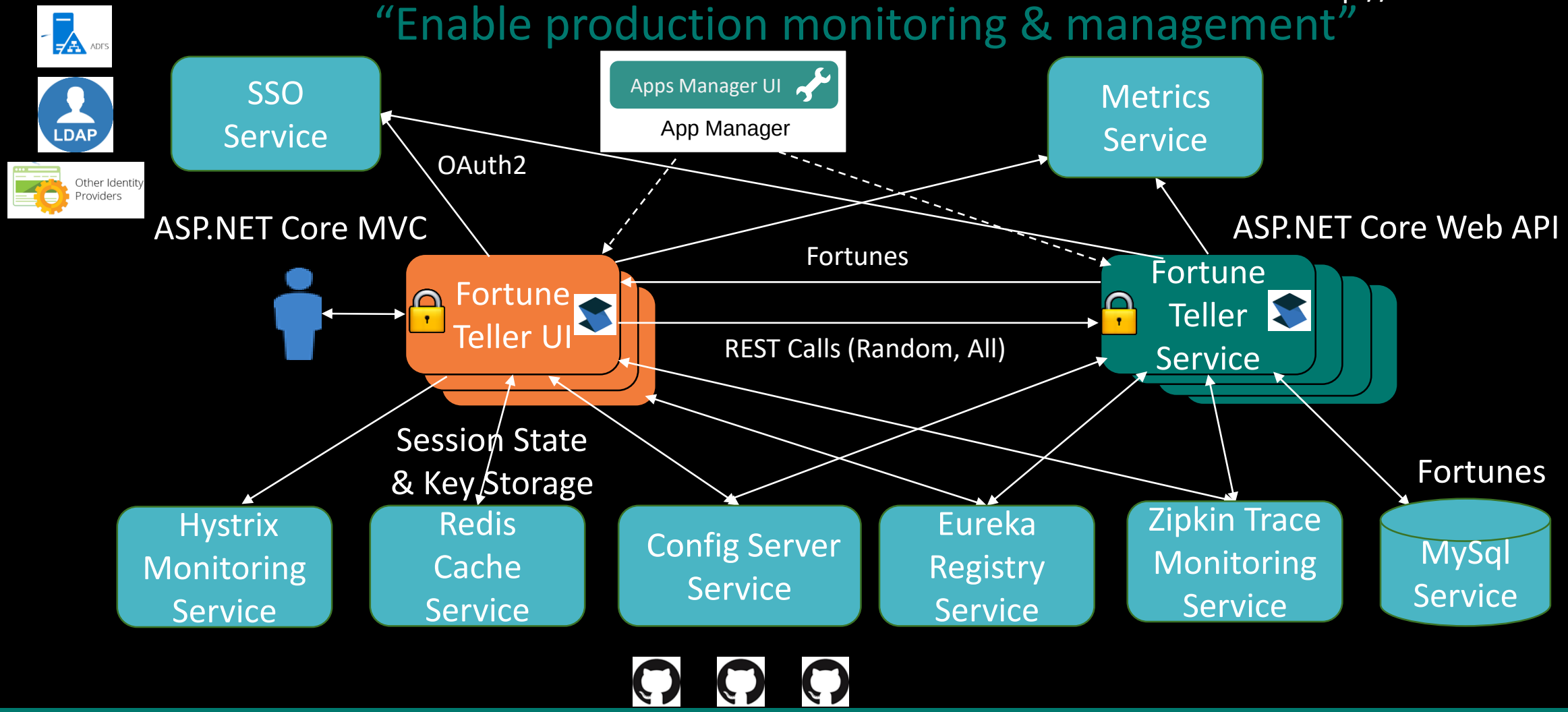
- Enables using OAuth2 tokens issued by Cloud Foundry Security services to secure access to REST resources/endpoints
- Usage
  - Add Authentication and Authorization services to container
  - Add Authentication middleware to pipeline
  - Apply Authorization attributes
  - Implement token forwarding as needed
  - On Cloud Foundry create & bind OAuth service instance for usage

To the Sample

Observability

Steeltoe
http://steeltoe.io

"Enable production monitoring & management"



Observability – Steeltoe Tools for Monitoring and Mgmt

- Management Endpoints – REST endpoints for monitoring & managing your apps
- Application Metrics – capture application and runtime performance data
- Distributed Tracing – capture and correlate application traces in a distributed system
- Hystrix Circuits – capture circuit status and performance data (i.e. external dependencies)

Observability – Steeltoe Management Endpoints

- Easily expose REST endpoints which integrate with Pivotal Apps Manager
  - /info - arbitrary app info, e.g. git build tag
  - /health – detailed application health information
  - /trace - circular buffer of last 100 http requests/responses
  - /loggers - show and dynamically modify log level of loggers
  - /mappings – show exposed routes (e.g. REST endpoints) in app
  - /dump - perform thread dump (.NET - Windows only)
  - /heapdump – return a gzip compressed dump file (.NET - Windows only)
- Additional endpoints accessible via REST calls (not integrated with Pivotal Apps Manager)
  - /env - expose app configuration key value pairs
  - /reload - cause applications configuration to be reloaded
  - /metrics - expose system and app metrics

Observability – Steeltoe Application Metrics

- Metrics exposed via HTTP Rest endpoint - /metrics
- CLR runtime metrics automatically captured
- App metrics automatically captured for common ingress and egress points (e.g. ASP.NET Core Host, MVC, Views & HttpClient)
- Optionally instrument your app yourself using OpenCensus APIs
- Optionally enable metrics exporting to backend metrics system
  - Metrics Forwarder for PCF supported
  - Others via OpenCensus

Observability – Steeltoe Distributed Tracing

- Add trace and span ids to log output – [appName, traceid, spanid, exported]
  - Facilitates log correlation
- Automatic instrumentation of common (.NET - ASP.NET & HTTPClient) ingress and egress points
- Automatic propagation of trace context to other processes
- Optionally instrument your app yourself using OpenCensus APIs
- Optionally enable trace exporting to backend tracing system
  - Zipkin
  - Others via OpenCensus

To the Sample

Questions