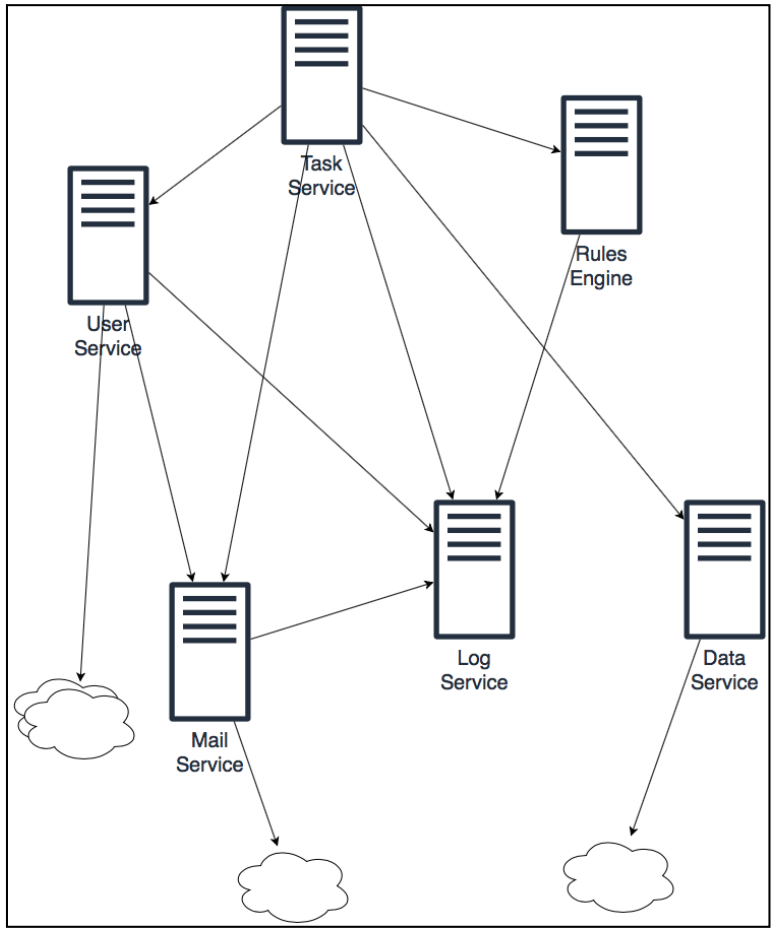
Testing HTTP Communication in your Applications

Nate Wells
Team Lead at Markel
Heartland Developers Conference
September 2019

Resiliency in a Microservice Landscape

Design for failure
and then test the failures



Consider this Sample Application:

Game Forecast

- The application has a local data store of college football games for this weekend.
- Each game has two teams, one stadium, and the date and time of kickoff
- Exposes an API to deliver the game properties plus the forecast for kickoff
  - Uses Google Places API to find the latitude and longitude of the stadium
  - NWS point API returns a list of NWS products for the given coordinates
  - The forecast for kickoff can be found by calling the NWS hourly forecast product

Google Places API

```
GET https://maps.googleapis.com
/maps/api/place/findplacefromtext/json
?input=Folsom+Field+Boulder,+CO
&inputtype=textquery
&fields=formatted_address,geometry,name
&key=your-google-api-key
```

Google Places API


```
{
  "candidates" : [
    {
     "formatted_address" : "2400 Colorado Ave, Boulder, CO 80302, USA",
     "geometry" : {
       "location" : {
             "lat" : 40.0094746,
             "lng" : -105.266905
       },
       "viewport" : { ... }
     },
     "name" : "Folsom Field"
    }
  ],
  "status" : "OK"
}
```

NWS Point API

```
GET https://api.weather.gov/points/40.0095,-105.2691
```

NWS Point API


```
{
...
 "id": "https://api.weather.gov/points/40.0095,-105.2691",
 "type": "Feature",
 "geometry": {
        "type": "Point",
        "coordinates": [
            -105.26909999999999,
            40.009500000000003
        ]
    },
 "forecast": "https://api.weather.gov/gridpoints/BOU/53,73/forecast",
 "forecastHourly": "https://api.weather.gov/gridpoints/BOU/53,73/forecast/hourly",
 "forecastGridData": "https://api.weather.gov/gridpoints/BOU/53,73",
 "observationStations": “https://api.weather.gov/gridpoints/BOU/53,73/stations”,
 "radarStation": “KFTG",
...
}
```

NWS Hourly Forecast API


```
{
...
    [ ...
            {
                "number": 90,
                "name": "",
                "startTime": "2019-09-07T13:00:00-06:00",
                "endTime": "2019-09-07T14:00:00-06:00",
                "isDaytime": true,
                "temperature": 84,
                "temperatureUnit": "F",
                "temperatureTrend": null,
                "windSpeed": "3 mph",
                "windDirection": "NW",
                "icon": "https://api.weather.gov/icons/land/day/tsra_hi?size=small",
                "shortForecast": "Chance Showers And Thunderstorms",
                "detailedForecast": ""
            },
    ... ]
...
}
```

.NET Core : Controller

```
[HttpGet("{id}")]
public async Task<ActionResult<Game>> GetGame(int id){
    var game = await _context.games.FindAsync( id );
    if( game == null ){
        return NotFound();
    }

    LocationSearchResult location = ( await _locationService.searchInCity(
        game.location.name,
        game.location.city,
        game.location.stateAbbreviation
    ) )[0];

    game.forecast = await _weatherService.GetForecast(
        location.latitude, location.longitude, game.kickoffTime
    );

    return game;
}
```

.NET Core : Weather Service

```
public class WeatherService {
    public HttpClient _client { get; }

    public WeatherService( HttpClient client ){
        client.BaseAddress = new System.Uri( "https://api.weather.gov" );
        client.DefaultRequestHeaders.Add(
            "user-agent", “dev.natewells.gameForecast-.netCore-2.2"
        );
        _client = client;
    }
    ...
}
```

.NET Core : Weather Service

```
public async Task<HourForecast> GetForecast(
    double latitude, double longitude, DateTime dateTime
){
    HourForecast hourForecast = new HourForecast();

    var pointResponse = await _client.GetAsync(
        String.Format( "/points/{0:F4},{1:F4}", latitude, longitude )
    );

    JObject pointInfo = JObject.Parse( await pointResponse.Content.ReadAsStringAsync() );

    string forecastUrl = pointInfo["properties"].Value<string>("forecastHourly");

    ...
```

.NET Core : Weather Service

```
    var forecastResponse = await _client.GetAsync(
        forecastUrl.Replace(_client.BaseAddress.ToString(), "")
    );

    string forecastContent = await forecastResponse.Content.ReadAsStringAsync();

    JObject forecastData = JObject.Parse( forecastContent );

    if( forecastData["properties"] != null && forecastData["properties"]["periods"] != null ){
        foreach( var p in forecastData["properties"]["periods"] ){
            if(  dateTime >= p.Value<DateTime>("startTime") &&
                 dateTime < p.Value<DateTime>("endTime") ){
               hourForecast.temperature = p.Value<int>("temperature");
               ...
            }
        }
    }
    return hourForecast;
}
```

Grails 2 : Controller

```
LocationSearchResult stadiumLocation = locationService.searchInCity(
  game.location.name, game.location.city, game.location.stateAbbreviation ).first()

HourForecast forecast = forecastService.getForecastForHour(
  stadiumLocation.latitude, stadiumLocation.longitude, game.kickoffTime )

[
        kickoff: game.kickoffTime.format('yyyy-MM-dd HH:mm:ssZ'),
        home: [ school: game.home.school, mascot: game.home.mascot ],
        visitor: [ school: game.visitor.school, mascot: game.visitor.mascot ],
        forecast: [
                temperature: forecast.temperature,
                windSpeed: forecast.windSpeed,
                windDirection: forecast.windDirection,
                weatherDescription: forecast.weatherDescription
        ]
]
```

Grails 2 : Weather Service

```
class WeatherService {

    def grailsApplication
    RESTClient client  = new RESTClient()

    HourForecast getForecastForHour( Double latitude, Double longitude, Date hour ) {
        HourForecast forecast = new HourForecast()
        String baseUrl = grailsApplication.config.gameForecast.nws.baseUrl
        client.url = baseUrl
```

Grails 2 : Weather Service

```
def pointResponse = client.get(
        path: "/points/${ latitude.round(4) },${ longitude.round(4) }",
        headers: [
                'user-agent': "dev.natewells.GameForecast-grails2.5.1"
        ],
        accept: ContentType.JSON
)

if( pointResponse.statusCode == 200 ){
    // parse the response JSON and grab the URL to the hourly forecast product.
    def jsonPointResponse = new JsonSlurper().parseText( pointResponse.contentAsString )
    String forecastPath = jsonPointResponse.properties.forecastHourly.replace( baseUrl, '' )
```

Grails 2 : Weather Service

```
def forecastResponse = client.get(
        path: forecastPath,
        headers: [
                'user-agent': "dev.natewells.GameForecast-grails2.5.1"
        ],
        accept: ContentType.JSON)

if( forecastResponse.statusCode == 200 ){
    def jsonForecastResponse = new JsonSlurper().parseText(
                forecastResponse.contentAsString )

    def forecastPeriod = jsonForecastResponse.properties.periods.find {
                Date.parse("yyyy-MM-dd'T'HH:mm:ssXXX", it.startTime) <= hour &&
                hour < Date.parse("yyyy-MM-dd'T'HH:mm:ssXXX", it.endTime)
    }
    if( forecastPeriod ){
        forecast.temperature = forecastPeriod.temperature
        ...
```

What to Test

- Happy Path
- Unsuccessful responses
- Incomplete data
- Changes to data structure
- Specific error phrases
- Slow responses
- Application code only

Example Test Scenarios: Positive & Negative

	200	401	4xx	5xx	Invalid Respone	Missing Data	Slow
Google	X	X	X	X	X	X	X
NWS Point	X		X	X	X	X	X
NWS Forecast	X		X	X	X	X	X

Testing Strategies

- The objective is to control the results of the HTTP calls your application is making.
- Options:
  - Replace the objects within your application that perform the HTTP calls with mock objects.
  - Replace the HTTP destination with one that you control.

.NET Core : Testing with Mock Class

```
public class WeatherServiceTests {

    private readonly ITestOutputHelper _output;
    private WeatherService _service;

    public WeatherServiceTests( ITestOutputHelper output ){
        _output = output;
        _service = new WeatherService( new HttpClient( new NwsMockHttpMessageHandler() ) );
    }
```

.NET Core : Testing with Mock Class

```
public void test_GetForecast(
    double latitude, double longitude,  string dateTimeString,
    int temp,  string windSpeed, string windDirection, string description,
    string errorMessage
){
    DateTime hour = DateTime.Parse( dateTimeString, ...);
    var task = _service.GetForecast( latitude, longitude, hour );
    task.Wait();
    var forecast = task.Result;

    if( errorMessage.Length > 0 ){
        Assert.Equal( errorMessage, forecast.errorMessage );
    } else {
        Assert.Equal( temp, forecast.temperature );
        Assert.Equal( windSpeed, forecast.windSpeed );
        Assert.Equal( windDirection, forecast.windDirection );
        Assert.Equal( description, forecast.weatherDescription );
    }
}
```

.NET Core : Testing with Mock Class

```
[Theory]
// happy path
[InlineData( 40.001,  -105.269, "2019-09-03T19:30:00-06:00", 75, "5 mph", "E", "Chance Showers And Thunderstorms", "" )]
// happy path; earliest possible time
[InlineData( 40.001,  -105.269, "2019-08-31T22:00:00-06:00", 71, "3 mph", "WNW", "Partly Cloudy", ""  )]
// happy path; latest possible time
[InlineData( 40.001,  -105.269, "2019-09-07T09:59:59-06:00", 67, "2 mph", "W", "Partly Sunny", ""  )]
// too far in the future
[InlineData( 40.001,  -105.269, "2019-09-07T10:00:00-06:00", 0, null, null, null, "Forecast not available for requested time."  )]
// in the past
[InlineData( 40.001,  -105.269, "2019-08-31T21:59:59-06:00", 0, null, null, null, "Forecast not available for requested time."  )]
// invalid location (Returns a 500)
[InlineData( -105.269, 40.001,  "2019-09-03T19:30:00-06:00", 0, null, null, null, "Unsupported location."  )]
// invalid location (Returns a 404)
[InlineData( 0,        0,       "2019-09-03T19:30:00-06:00", 0, null, null, null, "Unsupported location."  )]
// location without hourly forecasts
[InlineData( 10,       0,       "2019-09-03T19:30:00-06:00", 0, null, null, null, "Hourly forecast product not available."  )]
// hourly forecast call returns 404
[InlineData( 80,       0,       "2019-09-03T19:30:00-06:00", 0, null, null, null, "Error retrieving the forecast."  )]
// hourly forecast call returns 500
[InlineData( 81,       0,       "2019-09-03T19:30:00-06:00", 0, null, null, null, "Error retrieving the forecast."  )]
// hourly forecast call returns unexpected JSON
[InlineData( 82,       0,       "2019-09-03T19:30:00-06:00", 0, null, null, null, "Unexpected response from NWS." )]
public void test_GetForecast(
```

.NET Core : Testing with Mock Class

```
public class NwsMockHttpMessageHandler : HttpMessageHandler {
    protected override async Task<HttpResponseMessage> SendAsync(
       HttpRequestMessage request,
       CancellationToken cancellationToken
    ){
        var responseMessage = new HttpResponseMessage( );
        string responsePayload = "";
        responseMessage.StatusCode = HttpStatusCode.NotFound;

        if( request.RequestUri.AbsolutePath.StartsWith( "/points" ) ){
            responseMessage.StatusCode = HttpStatusCode.OK;
            responsePayload = pointResponse;
            switch( request.RequestUri.AbsolutePath["/points/".Length] ){
                case '-':
                    responseMessage.StatusCode = HttpStatusCode.InternalServerError;
                    break;
...
        responseMessage.Content = new StringContent( responsePayload );

        return await Task.FromResult(responseMessage);
    }
```

Grails 2 : Testing with Mock Class

```
@TestFor(ForecastService)
class ForecastServiceSpec extends Specification {

    @Unroll
    void "test getForecastForHour: #label"() {
        service.client = Mock( RESTClient )
        Date hour = Date.parse( 'yyyy-MM-dd HH:mm', timeString )

        when:
        def result = service.getForecastForHour( lat, lon, hour )
```

Grails 2 : Testing with Mock Class

```
then:
callCnt * service.client.get(_) >> { Map args ->
    Response response = Mock( Response )
    if( args.path?.startsWith( '/points/' ) ){
        switch( args.path.toString().charAt( '/points/'.size() ) ){
            case '-':
                response.getStatusCode() >> 404
                break
            default:
                response.getStatusCode() >> 200
                response.getContentAsString() >> mockPointResponse
                break
        }

result.errorMessage == expectedResult.errorMessage
result.temperature == expectedResult.temperature
```

Grails 2 : Testing with Mock Class

```
where:
label                | lat  | lon  | timeString          | callCnt | expectedResult
'bad location'       | -102 | 30   | '2019-09-03 14:30' | 1       | new HourForecast(errorMessage: "Invalid location.")
'yesterday'          | 30   | -102 | '2019-08-31 23:00' | 2       | new HourForecast(errorMessage: "Forecast not available.")
'waaay in the future' | 30   | -102 | '2019-09-30 14:30' | 2       | new HourForecast(errorMessage: "Forecast not available.")
'happy path'         | 30   | -102 | '2019-09-03 14:30' | 2       | new HourForecast(temperature: 81, windDirection: 'ENE',
'earliest happy path' | 30   | -102 | '2019-09-01 22:00' | 2       | new HourForecast(temperature: 78, windDirection: 'W',
'latest happy path'  | 30   | -102 | '2019-09-04 23:59' | 2       | new HourForecast(temperature: 70, windDirection: 'WSW',
'no hourly URL'      | 80   | -102 | '2019-09-03 12:30' | 1       | new HourForecast(errorMessage: 'Forecast details not
'hourly: 400'        | 81   | -102 | '2019-09-03 12:30' | 2       | new HourForecast(errorMessage: 'Forecast request failed.')
'hourly: bad data'   | 82   | -102 | '2019-09-03 12:30' | 2       | new HourForecast(errorMessage: 'Forecast not available.')
```

Taking Mocking Further

- Construct all of the mock listeners in a similar fashion.
- Register all of the mock listeners in a catalog.
- Implement a dispatcher to intercept HTTP calls, scan the catalog and send requests to the appropriate mock listener.
- Based on environment-specific configuration, inject the mock dispatcher into your services.

Creating a Mock API Application

- Pick your favorite lightweight web framework.
- Mock out the endpoints of the APIs that your application integrates to.
- Deploy your mock API somewhere secure.
- Use environment-specific configuration to point your application your mock API instead of the real APIs.

Takeaways

- Design for failure up front.
- Test failures in isolation.
- Develop a pattern to control the result of the HTTP calls being made.
- Abstraction can open up options for other patterns for resiliency and optimization.
- Establish these patterns and apply them to all of your applications.

Next Level

- Mountebank : OSS platform for developing API “doubles”
- Resiliency libraries for retry, fallback, circuit breakers, etc.:
  - Resilience4j (Java)
  - Polly (.NET)

References

- https://martinfowler.com/articles/microservices.html
- https://martinfowler.com/articles/microservice-testing
- https://touk.pl/blog/2013/01/03/using-wslite-in-practice/
- https://gingter.org/2018/07/26/how-to-mock-httpclient-in-your-net-c-unit-tests/
- https://github.com/natewells/http-testing-demos

Questions