The Microservice Checklist

Microservices provide the ability to breakdown complex systems into independent, and reusable, services. There are many hidden challenges in this approach and this Microservice checklist and the associated Microservice starter kit attempt to help you avoid these challenges!

01 SERVICE NAMING

All services should follow the same naming convention and use a DNS name to be addressed.

```
[service name]-service.[env].[domain].com
eg:
    album-service.dev.tiltwire.services
```

02 ENDPOINT NAMING

All endpoints should be resource based. Resources with more than one word should be hyphenated:

```
/albums
```

03 REST

All services should follow the REST standards (REST API URI Naming Conventions and Best Practices) and as such all endpoints should be resource based and not action based (eg: employees instead of add-employee).

04 STANDARD PLAYLOAD FORMAT

All request and response payloads must be in JSON format with property names specified in camelCase.

```
{
    “firstName”: “John”,
    “lastName”: “Smith”,
    “address”: {
        “street”: “21 King St.”,
        “city”: “New York”,
        “state”: “NY”,
        “zipCode”: “10042”
    }
}
```

05 CONSISTENT ERROR RESULTS

Errors should be handled using an appropriate HTTP Result Code as well as a simple payload body with more details. The simple body should be as follows:

```
{
    “datetime”: “2024-03-21”,
    “traceId”: “”,
    “status”: 500,
    “title”: “Internal Server Error”,
    “stackTrace”: “”
}
```

06 CENTRALIZED LOGGING

By default, Docker/K8s logs are written to the console inside each pod when running inside K8S. This is troublesome as it can require ssh access to the host in order to view the logs. Further, logs are isolated from each other making tracing calls across multiple services difficult and time consuming. The ELK stack solves this problem for us. All services should write all logs to Logstash which makes them available for searching/filtering in Kibana. This is a very powerful tool as it allows us to filter by date/time range, service name, transactionId and other things in a convenient and powerful UI.

07 OPENAPI (IN NON-PROD!)

Swagger is an excellent middleware that exposes a webpage that allows developers to fully interact with all endpoints exposed in a service. It also supports Authentication so a JWT token can be provided on guarded calls. Most languages provide an implementation of Swagger.

08 SERVICE DISCOVERY

A service registry is a central location for service discovery. It prevents services and apps from having to hard code or make assumptions about the locations of dependent URIs. All service dependencies should be retrieved, at startup, from the Service Registry by calling on of the endpoints available.

09 DB MIGRATIONS

When dealing with dozens of services, across multiple environments, it is critical that we reduce any and all manual intervention required to get a service up and running. As such, all services should use a framework to enable automated database migrations which will handle creating all required schema and any data population/manipulation to make the service function as expected once deployed. Some examples are Entity Framework (C#) and Flyway (Java).

10 HEALTH ENDPOINT

A healthy service is a happy service! A health endpoints ensures that service health can be easily checked by K8S or other systems.

```
eg:
https://some-service.dev.tiltwire.services/healthz
```

tiltwire

For the complete Microservice Starter Kit visit:

www.tiltwire.com/ms