Concept - API GW/Admin UI to manage the API GW - AWSIAPIM-646

APIoneer

AWSIAPIM-646 - Getting issue details... STATUS

Table of Contents

- Introduction
- Requirements
- Solution Approach
    - System Context
    - Container
        - Web Application
        - Micro Frontend App
        - API Services
        - Database
- Open Topics

Introduction

This technical concept describes the new **API GW Admin UI** system. Its purpose is to manage **API GW** and **AzS** instances for the different **SPR** environments from a central point. In a first step the focus is only on **SPR** but later on also the various **MSG** instances should be manged with this system too.

Requirements

- One admin GUI for all environments
- Service List with information about service and versions deployed on environments
    - All business service should be included in the Admin GUI:
    - filter based on name, contact owner (input also as RegEx possible), authMethod, enabled, scope, Payload/Swagger validation active, Swagger via url/S3 bucket, swagger access status
    - API guidelines for preferred configuration (auth method, scopes, rate-limit) for exposing API. List of service which have exposed Open APIs so that the service owners can be contacted.
    - List of service which are secured by same scope.
    - Future topic: Extend the portal to include the services from MBB
    - columns: service name, service owner, environments/versions
    - selectable rows
        - actions: send mail to all selected service owners
        - templates for mail creation
    - dashboard
        - service count
        - interface count
        - Link to Grafana dashboard
        - Link to Kibana logs
        - Infra resource usages
    - parameters adjustments

- Authentication/Authorization - ISAM/Cloud IDP. Roles service owner/admin will be stored in IDP but roles - service mapping will be stored in API GW.
- AzS - Client list on the basis of scopes. Overview of clients and service mappings. Network of permission with client and services. In client registration apart from scope take service/apps as well which will use this client.
- AzS - Overview of clients are enabled/disabled.
- AzS - List of clients are secured by basic authentication on QA/Live.
- AzS - Inform services owner if they are not following API guideline.
- AzS -> list of clients for services owners who are using their services so that for any breaking change can be allowed.
- AzS -> Different roles for Admin GUI. Service owner will only see list of client who is using their service. Admin role for all use cases.
- AzS - Approval of client configurations
- Differ tool web GUI link on admin portal. As a first step fetch the excel and show on the Admin Gui.
- Overview of certificates which will expired in coming days (1 month)
- Provision to update the scope with which the services are secured through Admin GUI. Let's say if a particular service is secured via a scope A and later it should be secured via B this can be done by API GW admin through Admin GUI

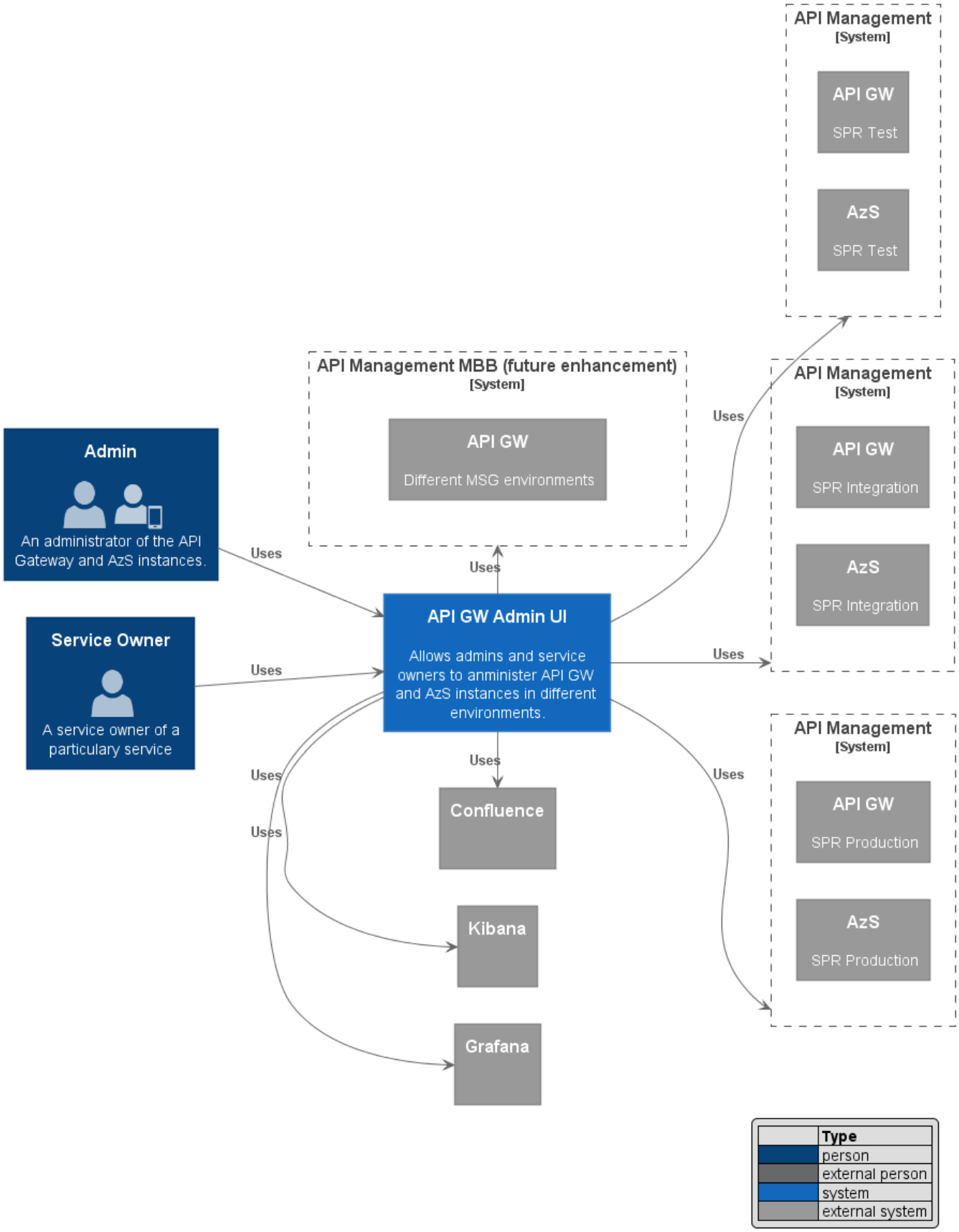
Solution Approach

The solution approach describes now how the different requirements and use cases will be supported by the new **API GW Admin UI** system from a technical perspective. The system context and the container diagram will illustrate the approach in more detail.

System Context

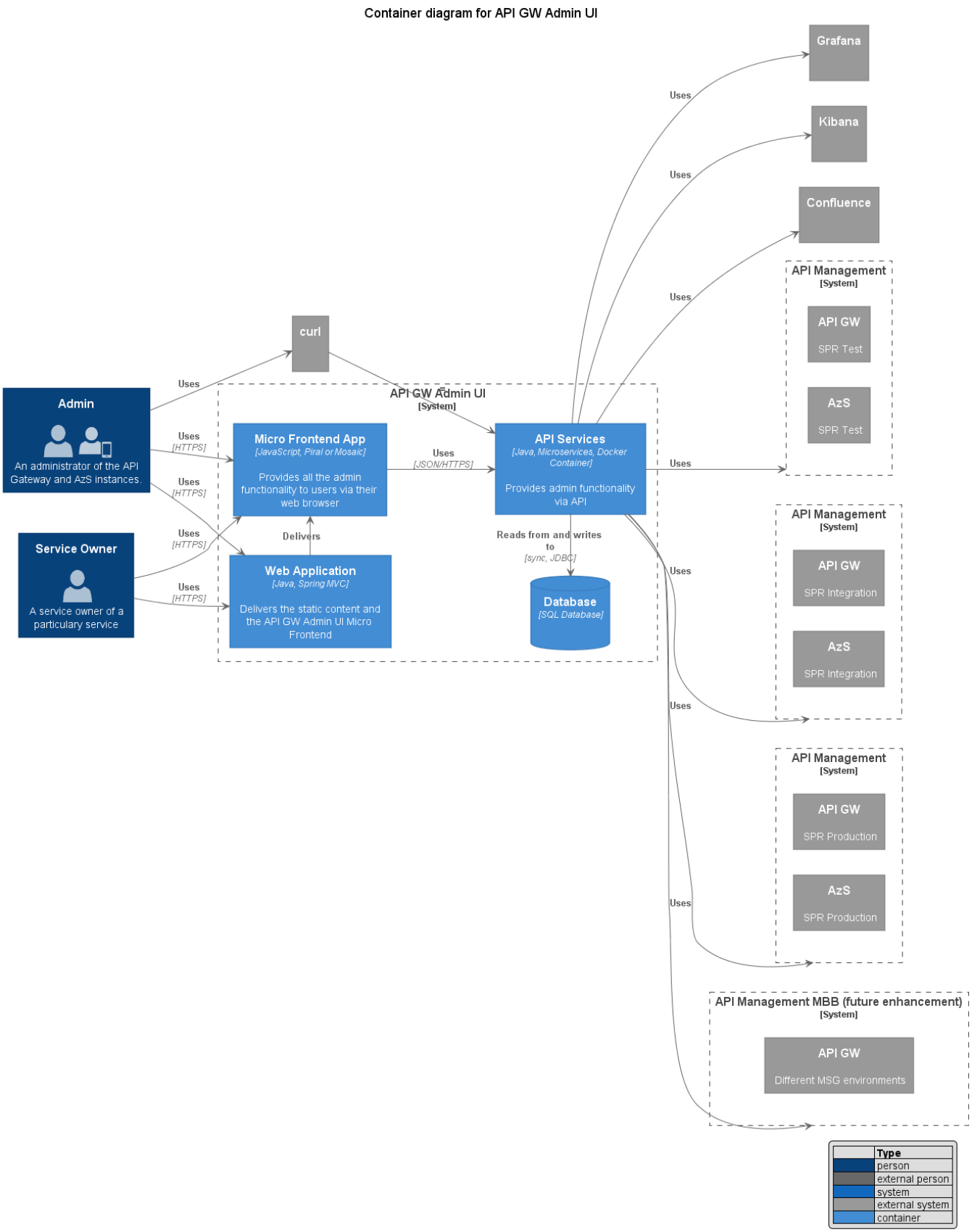
The **System Context** diagram provides a starting point, showing how the software system in scope fits into the world around it. The **API GW Admin UI** system will use and mange the API management systems on different environments (in a first step only **SPR** environments). There are two types of users foreseen (**Admin**, **Service Owner**) with different rights.

System Context diagram for API GW Admin UI



Container

The **Container** diagram zooms into the software system in scope, showing the high-level technical building blocks. A "container" is something like a server-side web application, single-page application, desktop application, mobile app, database schema, file system, etc. Essentially, a container is a separately runnable/deployable unit (e.g. a separate process space) that executes code or stores data. The mentioned **container** can run as a **Docker container** but this is not a must.



The **API GW Admin UI** system will consists of the following main components:

- Web Application
- Micro Frontend App
- API Services
- Database

All components will use Docker and Kubernetes.

Web Application

This component serves only as an entry point. It handles authentication and authorization via delegation. Provides static content and delivers the Micro Frontend App.

Micro Frontend App

The "real" UI will be provided by the **Micro Frontend App**. Micro frontends bring the concept of microservices to the UI side. Rather than defining micro frontends in terms of specific technical approaches or implementation details, we instead place emphasis on the attributes that emerge and the benefits they give:

- Incremental upgrades
- Simple, decoupled codebases
- Independent deployment
- Autonomous teams

There are several micro frontend frameworks where currently Piral and Mosaic (Zalando) are in the top ten. The proposal here is to use Piral but this will be discussed.

API Services

The API Services are a set of **microservices** which provide one API for the complete admin functionality. This fact allows us in the future to use certain Admin UI functionality in scripts, CI/CD pipelines, etc. with a simple curl call. This is comparable to the API GW, which provides a REST API (**restman**).

Database

The database is the persistence store.

Open Topics