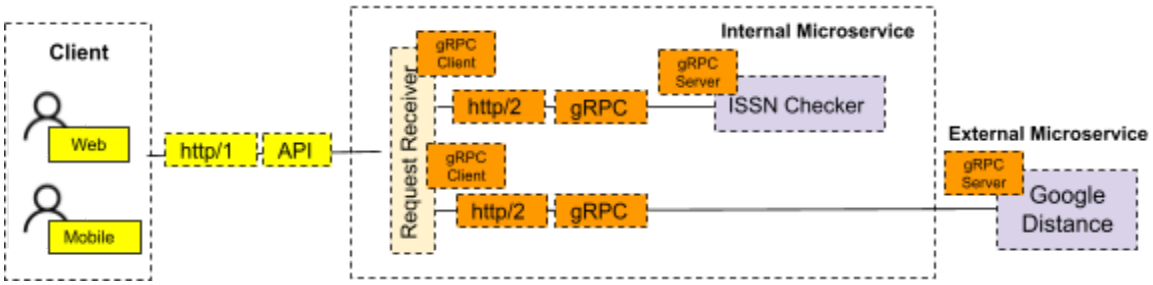
Development of Microservices: ISSN Checker and Distance Calculator

Microservice is a small independent software application or IoT device, provides services to other software applications or human clients through API or gRPC. Smart model is a generic model describing microservice provides. Smart feature is the function of the software application or IoT device accessed via the internet service. IoT device is a remotely controlled sensor which sends the latitude and longitude of the device, as well as its fuel level, so the functions of this IoT device (latitude, longitude of the device, and fuel level) are the smart features. An example of software is an ISSN checker software. Its smart feature is that it checks if ISSN is valid.
In this task we develop two microservices: In the diagram of the microservices, a microservices architecture is shown. The request from the client (web or mobile) reaches the Request Receiver API.



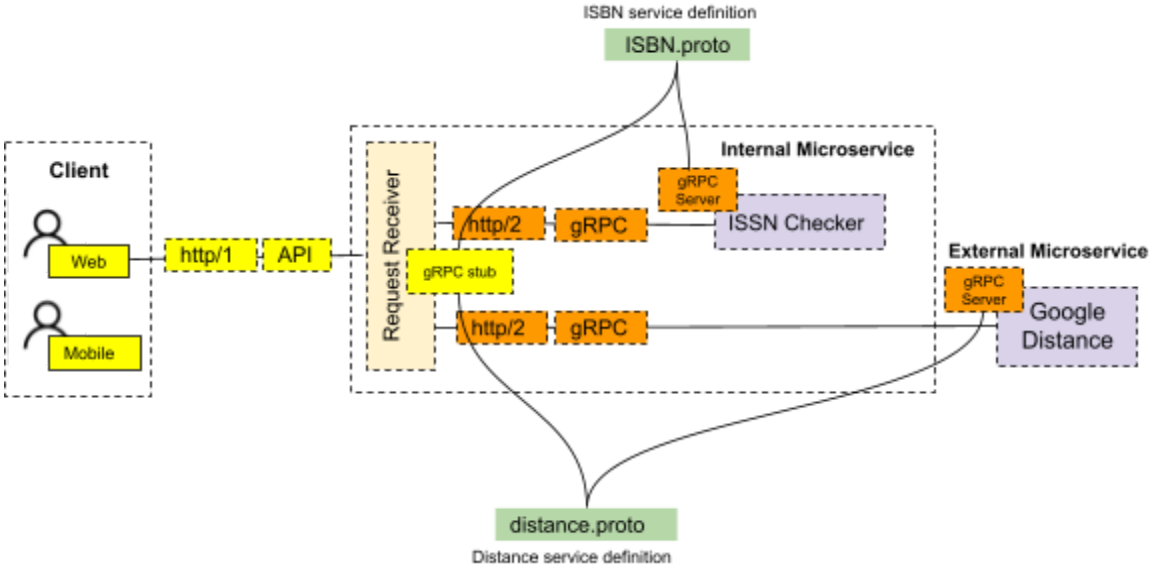
Task Schematic Diagrams

Then, the Request Receiver microservice calls the other two microservices the ISSN checker internally and Google Distance externally and aggregates the data to send back to the front end. As you can see, the communication between the request receiver and ISSN Checker Microservice and the Google Microservice happens through the gRPC protocol. If we use a normal REST API for communication between microservices it will tend to be slower and the client will experience more latency. We use gRPC between microservices, as it will be much faster and have low latency for the client.

gRPC is fast because of using HTTP/2 instead of HTTP/1.1 and protocol buffers instead of XML and JSON. gRPC uses HTTP/2 binary framing which is lightweight transport, and safer to decode compared to other text-based protocols.
The first step of building a gRPC connection is creating the service interface definition with the methods that are exposed by that service along with input parameters and return types.

The service definition is specified in the ISBN.proto file, which is used by both the ISBN checker microservice and Request Receiver microservice. And distance.proto used by Request Receiver

microservice and Google micro service as shown in below.



The microservice that provides the service is on Google cloud is the gRPC server and the client which requires data from the provider is the gRPC consumer is our Request Receiver microservice. With the service definition file, the source code can be generated using the protocol buffer compiler protoc. Also, on the server side, the server implements that service definition and runs a gRPC server to handle client calls. On the client side, we generate the client-side stub using the service definition. The client stub provides the same methods as the server and it converts them to remote function invocation network calls that go to the server side.
The microservice stores the smart features and services in persistent storage and retrieves them based on information such as name, identifier, type and category.

Installation Guide

First of all you can build the docker image and push it or you can pull the current version.

To build and push do the following:

Please make sure you build and push the latest changes to docker. for doing that please follow below steps:

Docker build -f "path to docker file" -t smartapi "c:\path\to\directory of project"

Then tag your build: docker image tag smartapi:latest [docker hub]:[your tag]

Then push it: docker push smartapi/smartapi:[your tag]

To pull and run the current version simply use this command:

Docker run -d –-name microservicesmartapi -p 8080:80 pooryabd/microservicesmartapi

Docker run -d –-name microservicesmartservicesrequestreceiver -p 7036:5036 pooryabd/microservicesmartservicesrequestreceiver

Docker run -d –-name microservicesmartservicesgoogledistance -p 7094:5094 pooryabd/microservicesmartservicesgoogledistance

Docker run -d –-name microservicesmartservicesissnchecker -p 7290:5290 pooryabd/microservicesmartservicesissnchecker

Deploy the container object to some sort of cloud Kubernetes, like Azure

Go to the Azure portal (AKS) and create a Kubernetes cluster

Install Azure CLI

Once Azure CLI is ready, run “az –version” at a command line. You should see an output.

Connect to the Cluster, please install Kubernetes CLI as well.

Once it’s installed, run “kubectl version –client” at your command line. You should see the output.

Now that you’ve created an AKS Kubernetes cluster, you need to connect your kubectl client to it. There is one more Azure-specific step to do this:

az aks get-credentials --resource-group apress-rg --name AksCluster Merged "AksCluster" as current context in C:\Users\myusername.kube\config

Define Kubernetes Objects (This step has already done).

Deploy to the Cluster:

kubectl apply -f app.yaml [your address] created

Please notice to change the google API keys in appsetting file.

After successful installation, open the browser and go to the address of API (https://+:80/swagger/index.html), here you will see two get methods

first one is Get Distance

second one is CheckISSNCode

you can see all details about the request and response here.