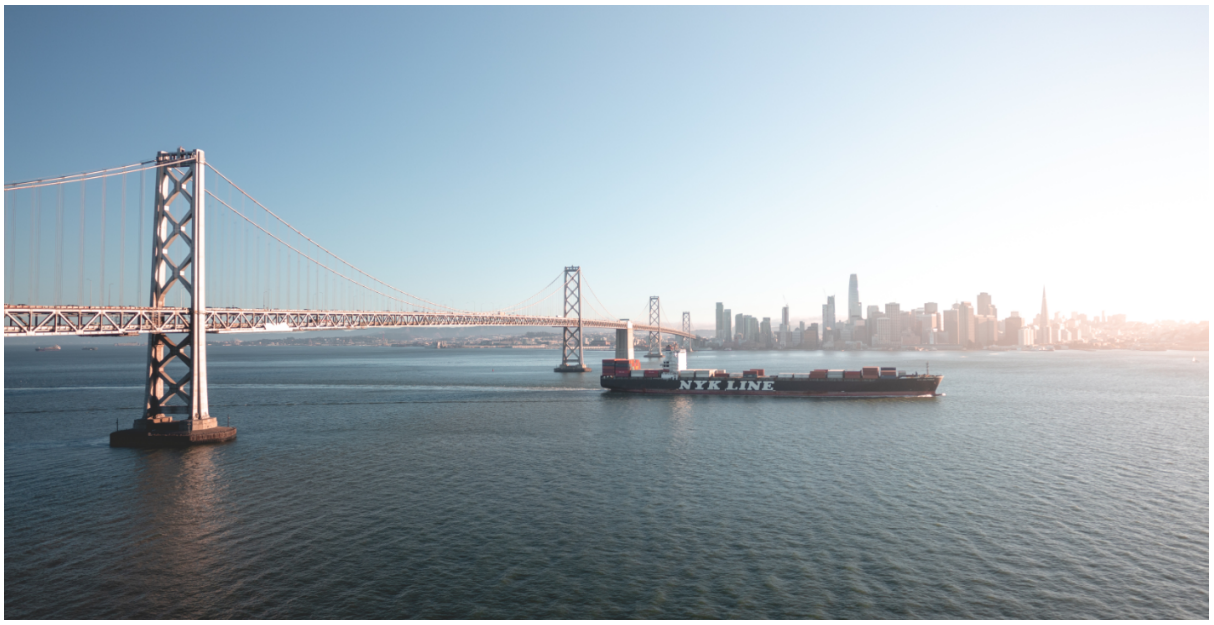
PORTCHAIN

Coding challenge:

Import and analyse container vessel schedules

Context about the shipping industry

90% of all manufactured goods in the world - worth $15 trillion - are moved through a container. The backbone of this global trade is made up of two main types of companies: Carriers and Terminal Operators.



Carriers are also known as shipping lines, and they operate the 6,000 container vessels sailing around the world at any given time between the different terminals. The vessels come in all sizes, with the biggest ones being 400m long, 60m wide and the capacity to carry more than 23,000 containers.

Terminal Operators operate the 1,200 terminals that exist globally. A terminal is a part of a port and they operate large cranes that move the containers between a vessel and a

parking space for the containers behind the cranes (called a Yard). Loaded and unloaded containers are continuously delivered and picked up by trucks, rail or other vessels, and transported either inland or to another port.

The vessels run on Schedules, which is the equivalent of a bus having a route with planned stops along the way, and when it will arrive at each stop. A stop for a vessel is called a Port Call, and globally there are 500,000 port calls annually.

Context about the data

- A port call is a vessel berthing at a container terminal to load and unload containers.
- A Port may have more than one container terminal. For simplicity in this exercise, the port calls only reference the port and not the actual container terminal.
- Sometimes a port call is present in a schedule but is temporarily skipped as a one-off event. This is called an "omitted" port call and is reflected through the `portCall.isOmitted` boolean field.
- The data in this challenge is real data that was gathered between Jan 1st 2019 and May 31st 2019.
- Although each vessel schedule is planned months in advance, delays and schedule changes happen frequently as the vessel approaches its different ports due to a range of reasons..
- A port call duration is the time during which the vessel stays berthed to load and unload its containers. This can be calculated by subtracting the arrival to the departure time:
  `portCallDuration = portCall.departure - portCall.arrival`

Your task

Your task is to build a tool to import vessel schedules from an external data source and display interesting statistics (defined below) about these schedules.

Step 1: Capture vessel schedules from the provided APIs

All available vessels can be listed here:
https://import-coding-challenge-api.portchain.com/api/v2/vessels

For each vessel, you can call the following API to get the full list of port calls and how they evolved with time (i.e., if they are omitted, delayed etc):
https://import-coding-challenge-api.portchain.com/api/v2/schedule/<vesselIMO>

For example, the following link requests the vessel schedule between January and May 2019 for the container ship MILANO BRIDGE:
https://import-coding-challenge-api.portchain.com/api/v2/schedule/9757187

Step 2: Analyse the data

From the data you have retrieved in Step 1, please now display the following information:

PORTCHAIN

- The five ports with the most port calls, and the number of port calls for each of these five ports.
- The five ports with the fewest port calls, and the number of port calls for each of these five ports.
- For each port, the percentiles of port call durations: 5th, 20th, 50th, 75th and 90th percentiles.

You can choose to display the data through a terminal/console or a web user depending on your own preference interface. Either case, make sure it's easily readable by the reviewer.

Guidelines

- Typescript is our preferred language but you can also use Javascript.
- Provide a solution to the steps above that you are proud of, and that you think reflect your skill sets. Pay special attention to the code organization and its readability.
- There is no need to store the data in a database or build a full webapp.
- Write tests for your code, including anything that may cover edge cases.
- Make the code easy to run (include a README file). You should write clear documentation that would allow anyone familiar with a shell to run your solution.
- Do not reinvent the wheel. Where possible, use existing tools and libraries.
- Don't hesitate to reach out if you have any questions along the way

Good luck!