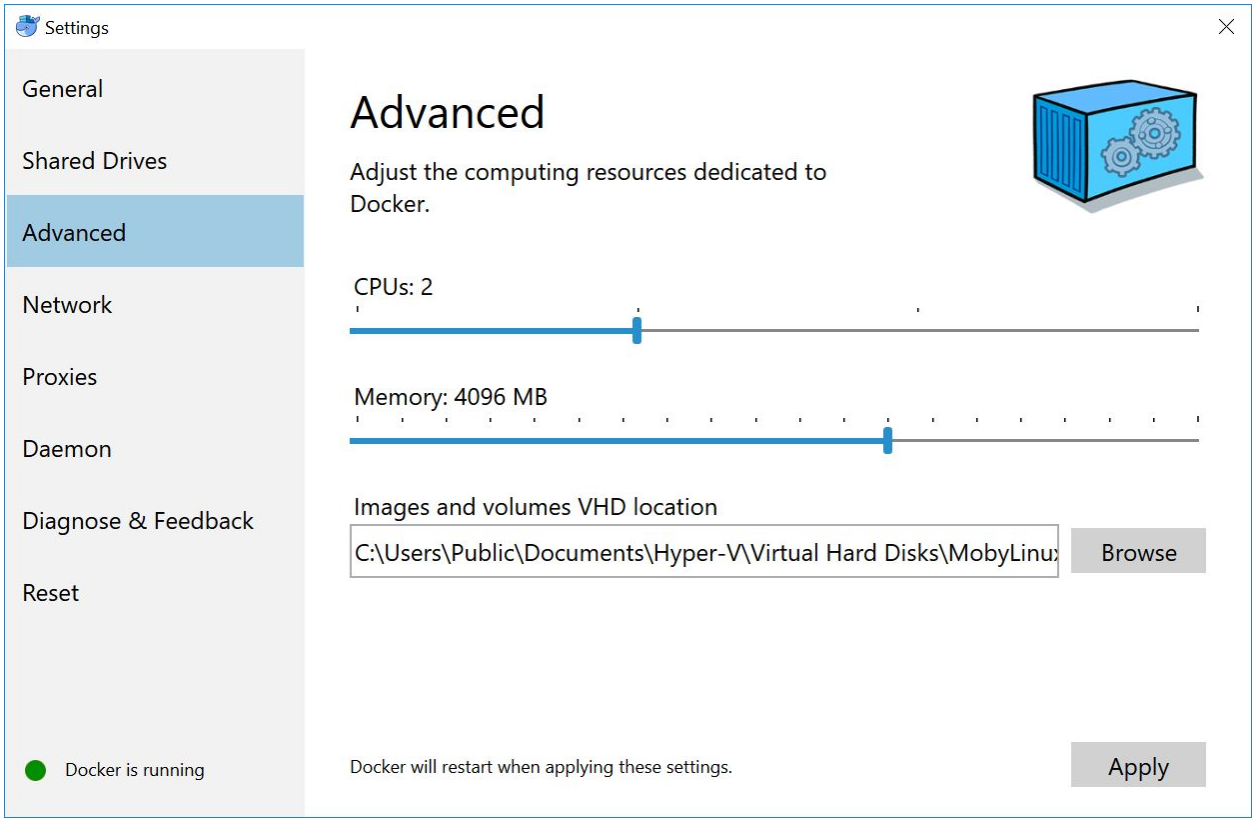
Growing Pelias in Containers

Setup

Install before the workshop

- Mapzen API Key
  - Follow these directions to create a free Mapzen Developer account and initialize a new API key
- command line shell
  - Windows: `cmd.exe`, PowerShell, git bash, or other
  - Mac OS: iTerm (bundled with OS) or iTerm2
- git
  - Windows
  - Linux and Mac OS
- gzip
  - Windows
  - Linux and Mac OS: bundled with OS
- docker & docker-compose (Community Edition)
  - Install
  - Increase the memory resources allocated for Docker
    - On MacOS, open the Docker→Preferences dialog and increase the **Memory** limit. Set it to **5.0 GB or more**.

- On Windows, right click on the docker whale icon in the taskbar. Select the **Settings** menu item and go to **Advanced** once the Settings dialog comes up. In the Advanced Settings tab, increase the memory limit to **5G or more**.



Clone and Download Images

1. Clone the pelias/dockerfile github repository as follows and we can begin by examining what has been obtained as part of the package. (Note: repository contents can also be found on the thumb-drive under **`<drive>/part1/dockerfiles`**) Once the repository in on your machine switch into the **`dockerfiles`** directory and browse its contents.

```
$ git clone https://github.com/pelias/dockerfiles.git dockerfiles
$ cd dockerfiles
$ ls .
```

   We can take a look at the docker-compose.yml file, which outlines all of the images that need to be built and the external data that needs to be mounted for each container.

2. We need to load the docker images for all of our components into the docker instance running on this machine. To download the prebuilt docker images from dockerhub you would use the docker-compose pull option. This should take ~10 minutes, but will vary depending on the internet connection.

Note: If you are unable to execute docker-compose commands without errors, it could mean the docker daemon isn't currently running. Please refer to the instructions for your OS to start up docker. (On MacOS and Windows it just means running the Docker application.)

```
$ docker-compose pull
```

After all the images have been built or loaded, we should be able to see a listing of them in docker.

```
$ docker images
```

Congratulations!

You have successfully prepared your machine for the upcoming workshop!

Introduction

What is Pelias?

Pelias is an open-source geocoding engine. It is data-agnostic and has several data importers available for prominent open data sources. Pelias supports the following key features of a geocoder:

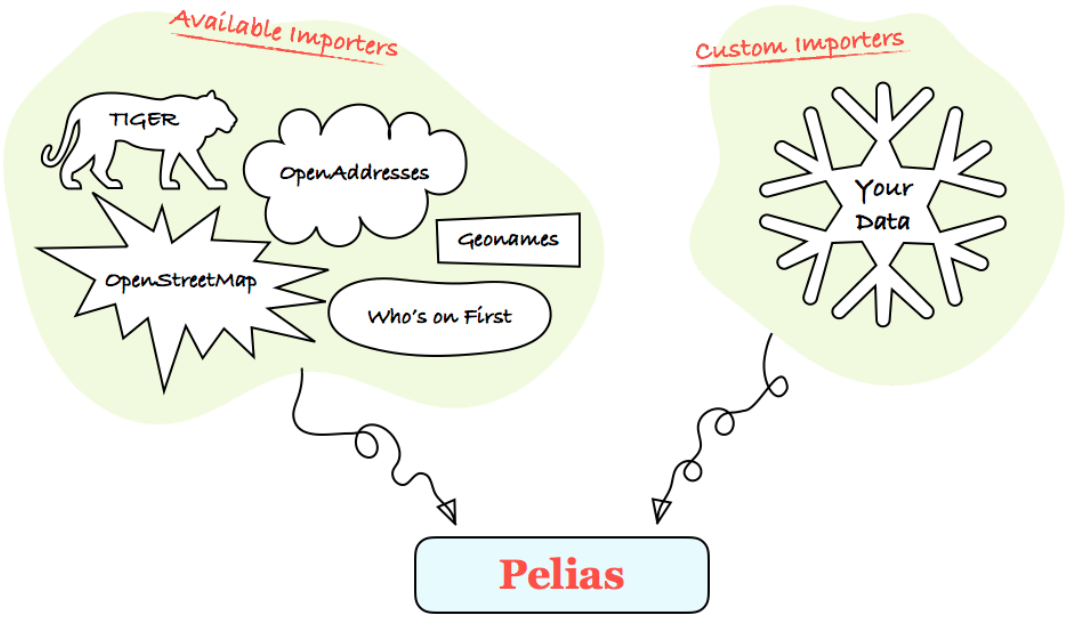
- forward geocoding, such as address and venue search
- autocomplete/typeahead, where suggestions are provided for incomplete input strings
- reverse geocoding, where a latitude/longitude pair are translated into the nearest address or the most granular administrative area surrounding the location
- filters and context parameters are available for versatile query customization

Pelias is inherently data-agnostic and allows users hosting their own instances to implement custom importers as needed. Custom data can be combined with other supported data sets to create powerful geocoding solutions.

What's the difference between Pelias and Mapzen Search?

Mapzen Search is a hosted instance of the Pelias geocoder maintained by the Mapzen Search team. It runs the `production` branches of all Pelias components and contains only open data. Mapzen Search provides global coverage and has a generous free tier to allow new users to experiment with the service as well as allow low-volume users to take advantage of the powerful global geocoder for free.

Which data sources are supported?



At this time importers are available for OpenStreetMap, OpenAddresses, Geonames, and Who's on First. Each data source represents a unique and critical piece of the geocoding puzzle.

- **OpenStreetMap**: crowdsourced venues, addresses, and polylines (road geometries)
- **OpenAddresses**: authoritative addresses
- **Who's on First**: administrative areas with full hierarchy and alternate names
- **TIGER**: US Census data with address ranges
- **Geonames**: venues and administrative areas *(soon to be deprecated)*

In addition to supporting all the abovementioned importers, Pelias also allows custom venue, address, and street data to be used to populate the index. Pelias is data-agnostic as long as the datasource provides the most basic information about each record.

What is Pelias comprised of?

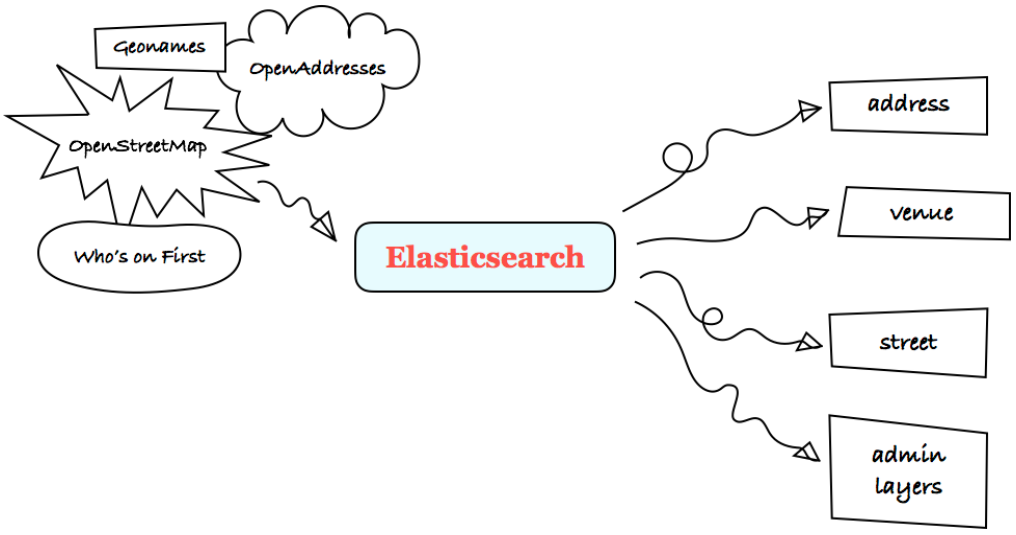
Dependencies

- node.js
- libpostal
- sqlite
- Elasticsearch

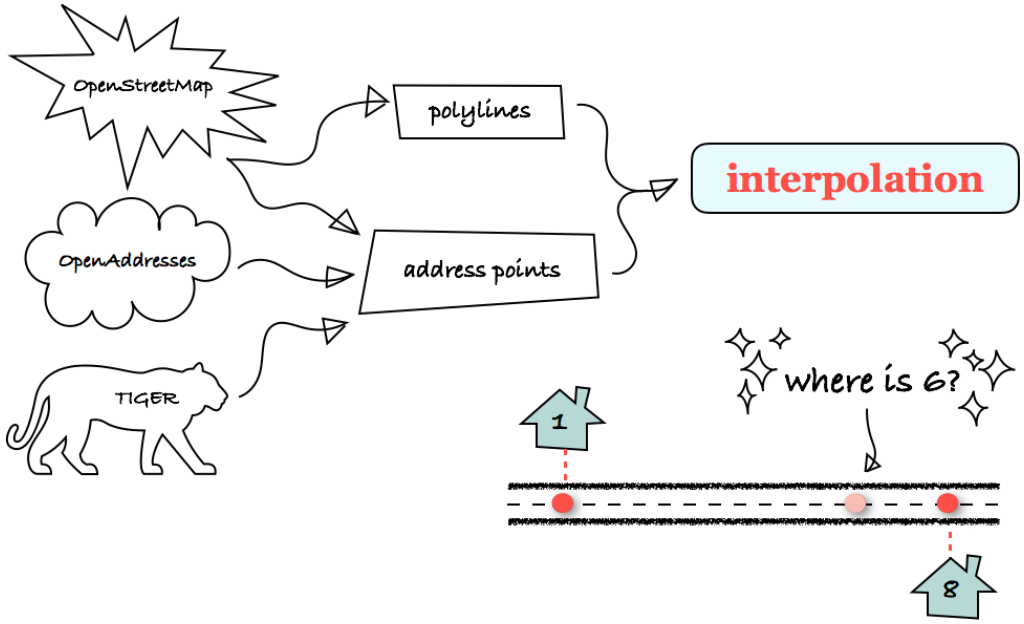
Long Running Services

- **API:** The brain of the Pelias engine. It is the entry-point and router for all user request and is responsible for all the following and more...
    - validating query parameters
    - parsing query text (using libpostal)
    - building queries to send to all other services
    - synthesizing results from all other services
    - decorating results with confidence scores and labels
    - removing duplicates where necessary

- **Elasticsearch:** Full text search engine that stores all venues, addresses, streets, and administrative areas as first class objects. All records have full administrative hierarchies so queries can limit results by parent information. Only points and centroids are stored in Elasticsearch; no polygons or lines.

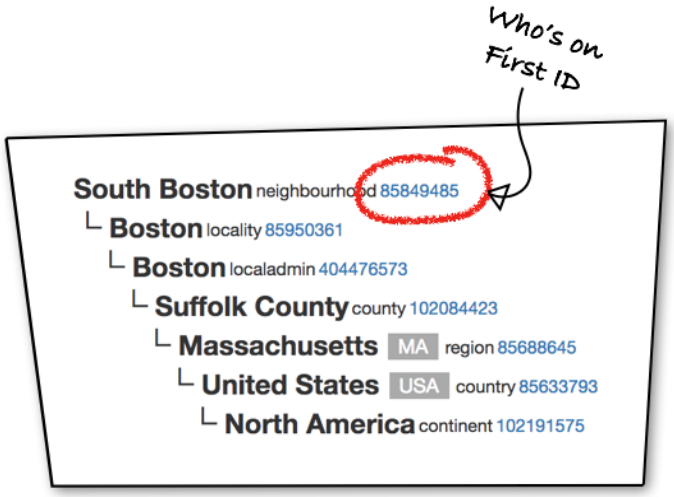


- **Interpolation:** Combines known address point data from OpenStreetMap, OpenAddresses, and TIGER, with road geometries from OpenStreetMap to allow house number estimation when an exact match is not found in the data.



- **PIP:** Stands for Point-in-Polygon which refers to the calculation of which polygons a given point lies within. This functionality is also often called Admin-Lookup. This service requires Who's on First data to be available at startup.

- **Placeholder:** Query parsing utility that preserves the hierarchical relationships between administrative records. It does not recognize venues, addresses, streets, or postalcodes, but does wonders for coarse records such as neighbourhood, locality, county, region, and country, just to name a few. Placeholder is also capable of finding a record by any of its alternate names, such as those in other languages or nicknames. This service relies on a custom built database derived from Who's on First data.



Data Importers

Each of the aforementioned services needs downloaded or prebuilt data in order to run. To generate this derived data Pelias provides data downloaders and importers. There are those that correspond directly to a particular data source and other that correspond to a particular service and combine several data sources into a single derived database.

- **whosonfirst:** [target: Elasticsearch] Responsible for downloading Who's on First data and importing it into the Elasticsearch index.
- **openstreetmap:** [target: Elasticsearch] Responsible for downloading OpenStreetMap data and importing only addresses and venues into the Elasticsearch index. Each record is enriched with a full admin hierarchy.
- **polylines:** [target: Elasticsearch] Responsible for extracting road geometries in the form of polylines from previously downloaded OpenStreetMap data and importing the centroids of all the roads into the Elasticsearch index. Each record is enriched with a full admin hierarchy.
- **openaddresses**: [target: Elasticsearch] Responsible for downloading OpenAddresses data and importing it into the Elasticsearch index. Each record is enriched with a full admin hierarchy.
- **interpolation:** [target: Interpolation Service] Responsible for synthesizing address point data from OpenStreetMap, OpenAddresses, and TIGER, with road network data from OpenStreetMap to allow for missing house number location estimation.
- **placeholder:** [target: Placeholder Service] Responsible for manipulating the previously downloaded Who's on First data and generating a sqlite database to be used by the Placeholder Service.

Local Pelias for Local Coverage

City Search

3. Let's clear out the current contents of the pelias.json file and start fresh with a very minimal setup. Using your favorite text editor, open up the **pelias.json** file, which can be found in the root directory of the dockerfile repository. Delete its contents and replace with the text inside the box below.

    - Be sure to insert your Mapzen API Key in the "api_key" property. To generate a key, if you haven't already, follow these instructions.
    - We also need to determine the Who's on First ID of the region by which we want to limit the data download. We can find the ID by using Mapzen Search Explorer or browsing the Who's on First Spelunker. We're going to search for the locality (aka city) of *Boston, MA* and use the source id in the results in our pelias.json configuration file.

Filename: pelias.json

```
{
  "esclient": {
    "hosts": [{ "host": "elasticsearch" }]
  },
  "api": {
    "textAnalyzer": "libpostal",
    "services": {
      "placeholder": {
        "url": "http://placeholder:4100"
      }
    }
  },
  "imports": {
    "whosonfirst": {
      "datapath": "/data/whosonfirst",
      "importVenues": false,
      "importPostalcodes": true,
      "importPlace": "85950361",
      "api_key": "your-mapzen-api-key"
    }
  }
}
```

4. Create a data directory that will have enough space for all the downloaded data and build artifacts. Plan for at least 1GB to be safe. Once the directory is created we will need to edit the **.env** file to specify the DATA_DIR environment variable.

```
$ mkdir pelias_data
```

Filename: .env

```
DATA_DIR=./pelias_data
```

5. Let's download the Who's on First data for the City of Boston. The configuration we specified in pelias.json earlier will be injected into this command.

   Note: If you receive an error like `'error: [download_data_filtered] 403'` then your api_key credentials are not valid.

```
$ docker-compose run --rm whosonfirst npm run download
```

6. Next we need to build the placeholder database as follows. Once the build is finished, take a look at the mounted shared data directory and see that it now contains whosonfirst and placeholder directories.

```
$ docker-compose run --rm placeholder npm run extract
$ docker-compose run --rm placeholder npm run build
$ ls ./pelias_data
```

7. Now we are ready to start up the placeholder service.

```
$ docker-compose up -d placeholder
```

8. Once placeholder is up and running we can test it by sending it some queries or check out its demo interface via http://localhost:4100/demo/#eng. *(If you can't test these queries in a browser, use cURL from the command line.)*
   - http://localhost:4100/parser/search?text=South+Boston+Waterfront

9. Note that we first need to startup all the services api depends upon. Because the api always expects Elasticsearch to be present, we need to start the elasticsearch container and create the Pelias schema, which tells Elasticsearch what records in the Pelias index will look like and how to analyze them properly.

```
$ docker-compose up -d elasticsearch
$ curl 'http://localhost:9200' # test to make sure ES is running
$ docker-compose run --rm schema npm run create_index
```

10. Finally the API container can be started.

```
$ docker-compose up -d api
```

11. This is a good time to see what containers are currently running in docker-compose to ensure that everything went smoothly. We should see pelias_api, pelias_placeholder, and pelias_elasticsearch with **Up** state, and pelias_baseimage and pelias_libpostal_baseimage in **Exit 0** state.

```
$ docker-compose ps
```

12. Let's try a few queries in our brand new Pelias API!

> **Tip:** You can install a plug-in for your browser to display JSON in a formatted manner. You can search the web store for your browser to find and install applicable products.
> Similarly, if using cURL to test the queries you can pipe your results to a utility like **jq**.

- http://localhost:4000/v1/search?text=boston
- http://localhost:4000/v1/search?text=Бостон

Coarse Reverse

We now have a forward geocoding of administrative areas, but we don't have a way to reverse the query. This means we aren't able to perform coarse reverse queries in order to determine the administrative hierarchy at a given location. Let's make that possible by bringing the PIP service container online!

1. First things first: let's update our pelias.json configuration to include pip service.

Filename: pelias.json

```
{
  "esclient": {
    "hosts": [{ "host": "elasticsearch" }]
  },
  "api": {
    "textAnalyzer": "libpostal",
    "services": {
      "placeholder": {
        "url": "http://placeholder:4100"
      },
      "pip": {
        "url": "http://pip-service:4200"
      }
    }
  },
  "imports": {
    "whosonfirst": {
      "datapath": "/data/whosonfirst",
      "importVenues": false,
      "importPostalcodes": true,
      "importPlace": "85950361",
      "api_key": "your-mapzen-api-key"
    }
  }
}
```

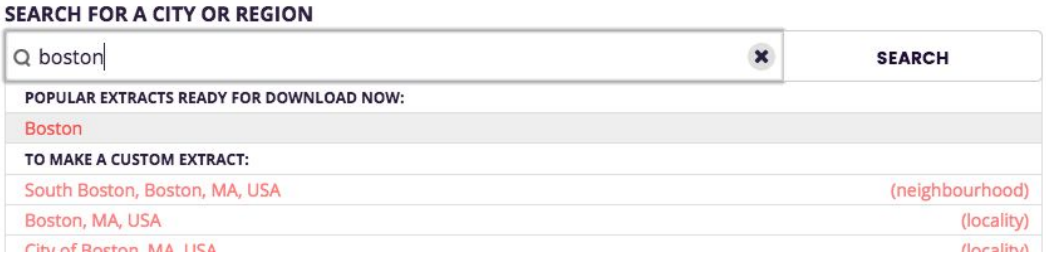
2. Now we just need to start up the pip service, since it doesn't require any special build process and just uses the downloaded Who's on First data. As with interpolation, we'll need to restart the api service to get pip discovered.

```
$ docker-compose up -d pip-service
$ docker-compose restart api
```

3. Time for a test to show off the coarse reverse capabilities brought to us by the pip service.
    - http://localhost:4000/v1/reverse?point.lat=42.351407&point.lon=-71.040492&layers=coarse

Venue Search

1. Venue data for Pelias comes from OpenStreetMap. Since we are only interested in the Boston region it makes sense to find an extract of the OpenStreetMap planet files containing only the data of interest. We can do so by referring to the [Mapzen Metro Extracts](). Once we've found the extract we're interested in we can grab the PBF file link under **Raw OpenStreetMap Data Sets,** for example https://s3.amazonaws.com/metro-extracts.mapzen.com/boston_massachusetts.osm.pbf. We will need to use this link in our configuration for the openstreetmap importer.



Add the openstreetmap importer configuration to pelias.json as follows. The highlighted parts are the additions and everything else should already be there from our previous steps.

Filename: pelias.json

```
{
  "esclient": {
    "hosts": [{ "host": "elasticsearch" }]
  },
  "api": {
    "textAnalyzer": "libpostal",
    "services": {
      "placeholder": {
        "url": "http://placeholder:4100"
      },
      "pip": {
        "url": "http://pip-service:4200"
      }
    }
  },
  "imports": {
    "whosonfirst": {
      "datapath": "/data/whosonfirst",
      "importVenues": false,
      "importPostalcodes": true,
      "importPlace": "85950361",
      "api_key": "your-mapzen-api-key"
    },
    "openstreetmap": {
      "download": [
        {
          "sourceURL":
"https://s3.amazonaws.com/metro-extracts.mapzen.com/boston_massachusetts.osm.pbf"
        }
      ],
      "leveldbpath": "/tmp",
      "datapath": "/data/openstreetmap",
      "import": [{
        "filename": "boston_massachusetts.osm.pbf"
      }]
    }
  }
}
```

2. Run the command to download the openstreetmap data.

```
$ docker-compose run --rm openstreetmap npm run download
```

3. Start the openstreetmap import process. This will take a few minutes to complete.

```
$ docker-compose run --rm openstreetmap npm start
```

4. We are now ready to try searching for venues. Here are some example queries that should be working.
    - http://localhost:4000/v1/search?text=YMCA

- http://localhost:4000/v1/search?text=public+library
- http://localhost:4000/v1/reverse?point.lat=42.351407&point.lon=-71.040492&size=1

Address Search

1. In order to bring in comprehensive address data for the City of Boston, we'll need to visit the OpenAddresses site and determine which of the sources we're interested in. For the purposes of the workshop, we're going to select the us/ma/city_of_boston data set.

2. Let's add the relevant configuration to pelias.json. As before the highlighted part is new, and everything else we've already seen.

Filename: pelias.json

```
{
  "esclient": {
    "hosts": [{ "host": "elasticsearch" }]
  },
  "api": {
    "textAnalyzer": "libpostal",
    "services": {
      "placeholder": {
        "url": "http://placeholder:4100"
      },
      "pip": {
        "url": "http://pip-service:4200"
      }
    }
  },
  "imports": {
    "whosonfirst": {
      "datapath": "/data/whosonfirst",
      "importVenues": false,
      "importPostalcodes": true,
      "importPlace": "85950361",
      "api_key": "your-mapzen-api-key"
    },
    "openstreetmap": {
      "download": [
        {
          "sourceURL":
"https://s3.amazonaws.com/metro-extracts.mapzen.com/boston_massachusetts.osm.pbf"
        }
      ],
      "leveldbpath": "/tmp",
      "datapath": "/data/openstreetmap",
      "import": [{
        "filename": "boston_massachusetts.osm.pbf"
      }]
    },
    "openaddresses": {
      "datapath": "/data/openaddresses",
      "files": [ "us/ma/city_of_boston.csv" ]
    }
  }
}
```

3. We are ready to run the openaddresses download and build commands. The import process will take a few minutes to run.

```
$ docker-compose run --rm openaddresses npm run download
$ docker-compose run --rm openaddresses npm start
```

4. Once the import process is complete our API is ready to search for addresses. Let's take it for a spin!
    - http://localhost:4000/v1/search?text=1+Seaport+Lane,+Boston+MA
    - http://localhost:4000/v1/reverse?point.lat=42.351407&point.lon=-71.040492&size=1&layers=address

Street Search

1. What if we wanted to search for a street instead? They are currently not in our Elasticsearch index. Let's fix that. The good news is that we already have the raw data required to extract the streets from: OpenStreetMap! Let's add the necessary configuration to pelias.json to prepare the importer for action.

Filename: pelias.json

```
{
  "esclient": {
    "hosts": [{ "host": "elasticsearch" }]
  },
  "api": {
    "textAnalyzer": "libpostal",
    "services": {
      "placeholder": {
        "url": "http://placeholder:4100"
      },
      "pip": {
        "url": "http://pip-service:4200"
      }
    }
  },
  "imports": {
    "whosonfirst": {
      "datapath": "/data/whosonfirst",
      "importVenues": false,
      "importPostalcodes": true,
      "importPlace": "85950361",
      "api_key": "your-mapzen-api-key"
    },
    "openstreetmap": {
      "download": [
        {
          "sourceURL":
"https://s3.amazonaws.com/metro-extracts.mapzen.com/boston_massachusetts.osm.pbf"
        }
      ],
      "leveldbpath": "/tmp",
      "datapath": "/data/openstreetmap",
      "import": [{
        "filename": "boston_massachusetts.osm.pbf"
      }]
    },
    "openaddresses": {
      "datapath": "/data/openaddresses",
      "files": [ "us/ma/city_of_boston.csv" ]
    },
    "polyline": {
      "datapath": "/data/polylines",
      "files": [ "extract.0sv" ]
    }
  }
}
```

2. From the existing openstreetmap data, we will now derive a new data set we refer to as **polylines**. This command will run deceivingly quickly, but trust that the work happened and there's a new polylines directory with data under pelias_data.

```
$ docker-compose run --rm polylines bash ./docker_extract.sh
```

3. Once the polylines extract is ready, we need to start the polylines importer, just as we've done with all the others.

```
$ docker-compose run --rm polylines npm start
```

4. As before, we are now ready to run a few test queries to see the streets come back in results.
    - http://localhost:4000/v1/search?text=William+J+Day+Blvd&layers=street
    - http://localhost:4000/v1/search?text=100+Seaport+Lane,+Boston+MA

Bonus Round:

Interpolated Address Search

If you were paying attention, the last test query in the previous section attempted to find ***100 Seaport Lane, Boston MA*** which does not exist in our address data. In the case that Pelias isn't able to find an exact match for an address, it attempts to do the next best thing and find the street the address might be found on. That's a handy fallback mechanism, but in some cases we can do a lot better than a street centroid; we can estimate, or interpolate, the location of the address based on the geometry of the street and the other known address points along it. To get this functionality to work in our current system we will need to run the interpolation build, which requires some additional data from TIGER.

1. Let's first specify which TIGER region we are interested in downloading. Processing the entire US would be too time consuming. We can lookup the region code for Massachusetts in this table: **25** is our state_code. Time to add this to the pelias.json configuration file as we've already done many times today. *(Note that there are two new blocks in this configuration update! Don't miss the one at the end of the file.)*

Filename: pelias.json

```
{
  "esclient": {
    "hosts": [{ "host": "elasticsearch" }]
  },
  "api": {
    "textAnalyzer": "libpostal",
    "services": {
      "placeholder": {
        "url": "http://placeholder:4100"
      },
      "pip": {
        "url": "http://pip-service:4200"
      },
      "interpolation": {
        "url": "http://interpolation:4300"
      }
    }
  },
  "imports": {
    "whosonfirst": {
      "datapath": "/data/whosonfirst",
      "importVenues": false,
      "importPostalcodes": true,
      "importPlace": "85950361",
      "api_key": "your-mapzen-api-key"
    },
    "openstreetmap": {
      "download": [
        {
```

```
          "sourceURL":
"https://s3.amazonaws.com/metro-extracts.mapzen.com/boston_massachusetts.osm.pbf"
        }
      ],
      "leveldbpath": "/tmp",
      "datapath": "/data/openstreetmap",
      "import": [{
        "filename": "boston_massachusetts.osm.pbf"
      }]
    },
    "openaddresses": {
      "datapath": "/data/openaddresses",
      "files": [ "us/ma/city_of_boston.csv" ]
    },
    "polyline": {
      "datapath": "/data/polylines",
      "files": [ "extract.0sv" ]
    },
    "interpolation": {
      "download": {
        "tiger": {
          "datapath": "/data/tiger",
          "states": [
            {
              "state_code": 25
            }
          ]
        }
      }
    }
  }
}
```

2. Time to download the TIGER data.

```
$ docker-compose run --rm interpolation npm run download-tiger
```

3. With all the data at hand, we are ready to run the interpolation build. This will take roughly 5 minutes to complete.

```
$ docker-compose run --rm interpolation bash ./docker_build.sh
```

4. We will need to start up the interpolation service, and then restart the api service for the new configuration to take effect and for the interpolation service to be detected.

```
$ docker-compose up -d interpolation
$ docker-compose restart api
```

5. As tradition would have it, it's time to test the newly added functionality. We can also check out the web-app running on the interpolation service to get a better understanding

of how the interpolation engine works by going to this url http://localhost:4300/demo/#18/42.34822/-71.03920, and clicking on any street.

- http://localhost:4000/v1/search?text=100+Seaport+Lane,+Boston+MA

Congratulations!

You have successfully setup a full instance of the Pelias geocoder for the City of Boston! Thanks for making it this far.

You are awesome!

Teardown

1. To stop all the running containers you can run the docker-compose down command as follows.

```
$ docker-compose down
```