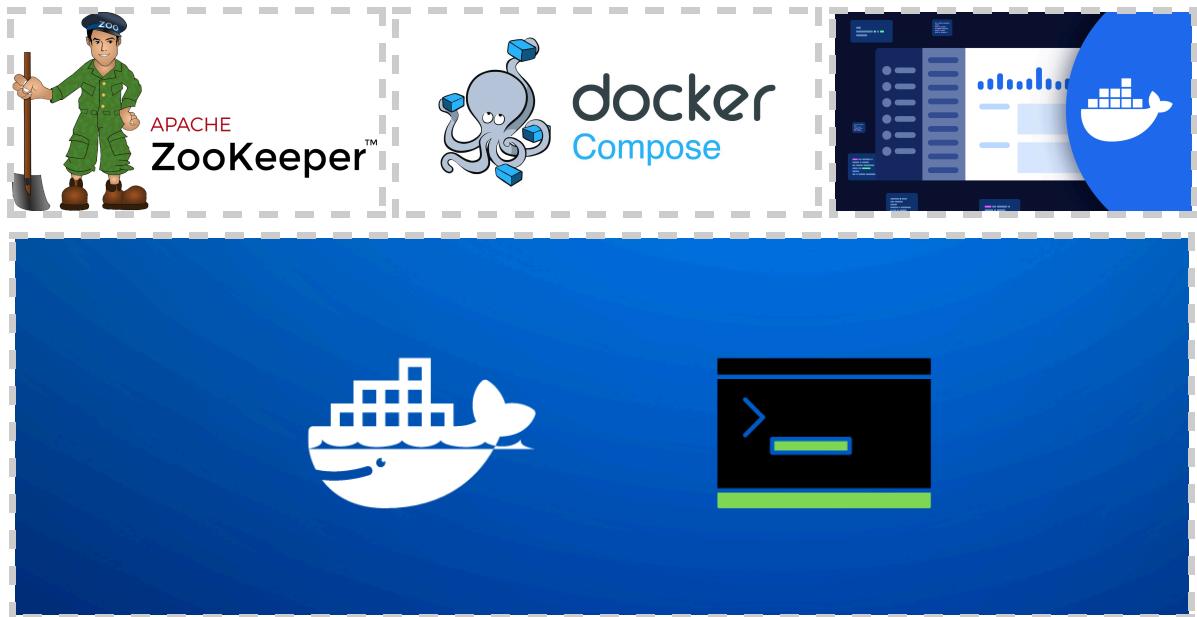


3. Container Debugging

This is the second out of the five guiding pdf files that aims to make it easier for the developer to get started with the project. In 4 sections, we will dive into how to configure and debug the containers so that you can view the status and outputs of the independent services



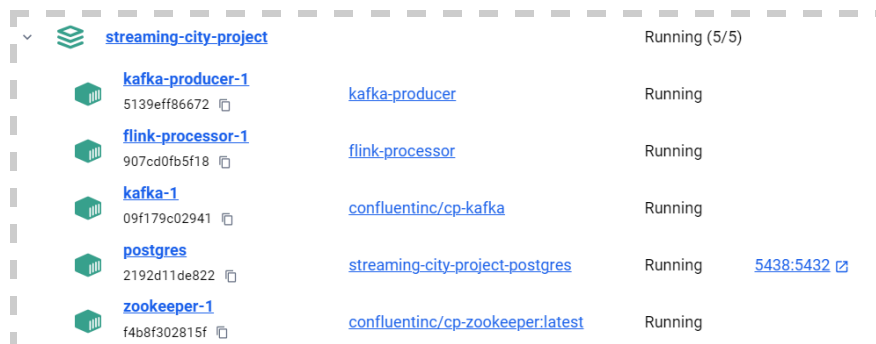
- 1) **Automation:** The information in this pdf file is based on the shell script `"/instructions-executions/executionary/3. container-debugging.sh".` Use this script as a reference to the explanations provided in this pdf. To be clear, you can just run `"3. container-debugging.sh"` and everything is taken care of automatically. With that said, this file is just providing elaborated explanations of the execution procedure. However, in the domain of container shell environments, the shell scripts are forced to enter new subshells and then run commands specific to those shells. This may be the cause for you not being able to run the script all at once. Instead, run the commands separately and make sure that you are in the correct shell. Inspecting the logs of containers may work by simply running the script, but to execute the PostgreSQL queries in the 'postgres' container, it is recommended to run each command manually.
- 2) **Configurations:** In the bash script, you are obligated to configure three variables: The container to debug, to display the container's logs or not, and to perform SQL queries

```
# The containers' names generated from "/application/docker-compose.yml". These can also be accessed through Docker Desktop
declare -a CONTAINERS=("postgres" "kafka-producer-1" "kafka-1" "zookeeper-1" "flink-processor-1")

## DEVELOPER CONFIGURATIONS ##

CONTAINER_TO_DEBUG=${CONTAINERS[1]} # Change the index to the corresponding container to debug here
CHECK_LOGS="True"
PERFORM_SQL_QUERIES="False"
```

Notice that the elements of `$CONTAINERS` are representative of the actual containers that are active. We can double check this via Docker Desktop, or the specified names in the `/application/docker-compose.yml` file:



Container Name	Image	Status	Ports
kafka-producer-1	kafka-producer	Running	
flink-processor-1	flink-processor	Running	
kafka-1	confluentinc/cp-kafka	Running	
postgres	streaming-city-project-postgres	Running	5438:5432
zookeeper-1	confluentinc/cp-zookeeper:latest	Running	

Note that if you want to query the SQL database, you must select the 'postgres' container by selecting the 0th index of `$CONTAINERS`. Now, the last step that remains is to provide a SQL query. You can either come up with your own or take inspiration from one of the associated .sql files:

```
3. city-queries.sql
3. container-debugging.sh
3. id-queries.sql
3. temperature-queries.sql
```

Once you know by what SQL query you want to filter the database, assign it as a string to the variable `$MY_QUERY`.

```
# Runs the specified .sql file based on the query type
runPostgresQueries() {

    connectToPostgresDatabase

    # Note: Change the command below to your desired query. A few example
    # queries can be found in the directory of this shell script. They are
    # contained in the .sql files that are tied to the 3rd tutorial.
    MY_QUERY="SELECT * FROM weather;"

    docker exec -it postgres psql -U postgres -d postgres -c "${MY_QUERY}"
}
```

3) Debug Containers: If we return to the configured settings in section 2, where we explicitly configured it to neglect SQL queries but to check the logs of the `"kafka-producer-1"` container. Running the script would give the following result in a successful case:


```
-- Select the instance with the highest id (the most recently inserted db-instance)
SELECT * FROM weather
WHERE id = (
    SELECT MAX(id) FROM weather
);
```

This would query the database and returns the following (at the time, 15670 db-instances were stored in the database):

```

root@DESKTOP-3529W411:/mnt/c/Ubuntu-Mounts/S...Streaming Pipeline/instructions-executions/executionary$ docker exec -it 2192d11de8224708c06fee2106eb3a33881d7502aaa471ded9fba26e9157a18e bash
root@2192d11de822:/# psql -U postgres -d postgres
psql (16.3 (Debian 16.3-1.pgdg120+1))
Type "help" for help.

postgres=# SELECT * FROM weather
postgres=# WHERE id = (
postgres=#     SELECT MAX(id) FROM weather
postgres=# );
 id | city | average_temperature
-----+-----+-----
15670 | Venice | 31
(1 row)

```

Another sql-file with query examples is "3. city-queries.sql". Lets now try this query:

```
-- Select only the temperature column from all instances that are tied to Rome
SELECT average_temperature FROM weather Where city LIKE 'Rome'
ORDER BY id ASC;
```

We receive the following output:

	average_temperature
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
34.299999999999976	34.3
	34.3
	34.3
	34.3
	34.3
	34.3
34.28928571428568	34.3
34.15084745762709	30.3
	30.3
	30.3
	30.3
	30.3
	30.3
	33.5
	30.3
33.25064935064937	30.3
	30.3
33.03086419753089	24.3
	24.3
	24.3
	24.3
	24.3

The last sql-example-file in which we will have a look at is “3. temperature-queries.sql”. In that file, one of its contained queries is:

```
-- Select every city with temperature greater than 35 in descending order
SELECT *
FROM weather
WHERE average_temperature > 35
ORDER BY average_temperature DESC;
```

Applying this to the ‘postgres’ container’s shell produces this:

id	city	average_temperature
15503	Dubai	40.4
15462	Dubai	40.4
14842	Dubai	40.4
14861	Dubai	40.4
14886	Dubai	40.4
14895	Dubai	40.4
14927	Dubai	40.4
14947	Dubai	40.4
14976	Dubai	40.4
14988	Dubai	40.4
15002	Dubai	40.4
15021	Dubai	40.4
16090	Dubai	40.4
16054	Dubai	40.4
16032	Dubai	40.4
16016	Dubai	40.4
15990	Dubai	40.4
15967	Dubai	40.4
15958	Dubai	40.4
15929	Dubai	40.4
14712	Dubai	40.4
14737	Dubai	40.4
14763	Dubai	40.4
14776	Dubai	40.4
15906	Dubai	40.4
14807	Dubai	40.4
15883	Dubai	40.4
14813	Dubai	40.4
15164	Dubai	40.4
15867	Dubai	40.4
15858	Dubai	40.4
15833	Dubai	40.4
15804	Dubai	40.4

- 4) **The next step:** In the next tutorial, we will be going through “4. real-time-charts.sh” which is predicated on this tutorial. Since we, in this file, learned about running the containers and debugging their real-time generated data, we will in the next tutorial make use of this data and visualize it.