Big Code for Live Traffic Generation

Semester Thesis
Author: Hande Harputluoglu
Tutor: Tobias Bühler, Roland Schmid
Supervisor: Prof. Dr. Laurent Vanbever

October to December 2022
Abstract

Github provides a big codebase of repositories which contain applications based on container technologies. Running these containers and capturing traffic can be an efficient source to generate network traces dataset. This dataset can be used to test network applications and systems. In this thesis, we propose a centralized architecture to capture traffic from repositories and process the results in an automatised, optimised and robust way which doesn’t require many user inputs. We prepare our system to be scalable and ready to support future ideas and implementations.
Contents

1 Introduction .............................................................. 1
  1.1 Motivation .......................................................... 1
  1.2 Task and Goals .................................................... 1
  1.3 Overview .......................................................... 1

2 Background and Related Work ...................................... 2
  2.1 Docker .............................................................. 2
  2.2 DYNAMO ........................................................... 3

3 Design ........................................................................ 4
  3.1 Motivations ........................................................... 4
  3.2 Centralized Architecture .......................................... 4
  3.3 Improvements on the Capture Process ........................ 6
    3.3.1 Subfolder Structure in Dataset ............................. 6
    3.3.2 Dataset Content Expansion ................................. 7
    3.3.3 Stability and Maintenance of Worker Machines ...... 8

4 Evaluation ................................................................... 10
  4.1 Discussions and Observations on Selected Repositories ... 10
  4.2 Statistical Analysis on Big Traffic Dataset .................. 11

5 Outlook and Summary .................................................. 15

References .................................................................... 16
Chapter 1

Introduction

1.1 Motivation

Many network researchers and operators use generated network traffic to test their network applications and systems. It is important to use traffic traces which are close to reality that users can fine tune according to their applications and network conditions. In DYNAMO, it is proposed to use container technologies from Github repositories to obtain network traffic traces [1]. This thesis is based on DYNAMO and aims to restructure its offline pipeline to have a more optimised and easy to work system while improving traffic capturing process.

1.2 Task and Goals

The main scope of this thesis is to build a centralized structure for DYNAMO’s offline phase while preparing system to be ready to support future ideas without rerunning the entire pipeline. Main tasks of the project are defined as following:

- Build a centralized structure which automatically distributes Github repositories over different machines, captures network traffic for given repositories and analyzes the captured traffic
- Expand data extraction in capture process
- Improve monitoring and maintenance of capture process on worker machines
- Inspect selected repositories manually to match docker containers with their captured traffic traces
- Analyze big traffic database to understand behaviours of collected traces

1.3 Overview

In section 2, we provide a background knowledge on Docker and DYNAMO. Section 3 presents the novelties we bring with the thesis: centralized architecture and improvements on the capture process. In section 4, we first do a manual inspection on selected repositories and discuss the general observations and behaviors on repositories. After, we do a statistical analysis on Big Traffic database to better understand our data and define the future steps. We summarize our work and introduce potential future work in section 5.
Chapter 2

Background and Related Work

2.1 Docker

Docker is an open source platform to build, deploy and manage applications which are based on containers. [2]

A docker container provides an isolated environment where an application can run quickly and reliably. This isolated environment is built up in a way to contain the application code and dependencies. It has its own resources, running as isolated processes in the user space. Many containers can be run on the same machine, without intersecting with each other. From the memory use, containers take up less space than virtual machines and can handle more applications. All these qualities make docker containers an attractive software development environment for developers.

A docker image is a description of the executable package of software which includes all the necessary information that application needed to be run: code, runtime, system tools, system libraries and applications. [3] In case we run docker images, they become docker containers. Docker images can be built from Dockerfile locally. Dockerfile defines the configurations and the code that image should be based on and allows the app environment to be reproduced.

Docker Hub is a public repository for container images. Users, developers and companies can build their own images and store it in Docker Hub. There are many docker applications which are based on official images of well known services. When we build a docker application, its images are either built locally or pulled from Docker Hub and then run. There are few types of subscriptions on Docker Hub. A personal account is limited to 200 image pulls per 6 hours while pro account allows users to pull 5,000 images per day. [4]

Docker compose is a tool to define and run multi container docker application. We need to have a docker compose file which defines the container properties and it is enough to use "docker-compose" command to run all the containers, as well as to stop and check their status.

A docker compose file defines the services, networks and volumes for a Docker application. Computing components of the applications are defined as services. Services communicate with each other though networks. [5] The main docker compose file has to be named within the group {compose.yaml, compose.yml, docker-compose.yaml, docker-compose.yml} to docker compose command to work efficiently.
2.2 DYNAMO

DYNAMO is a live application traffic generator. [1] It has two phases: an offline phase where Big Traffic database is created and a live traffic generation phase. Big Traffic database consists of captured traffic traces that are collected by running docker compose files from GitHub repositories. To obtain this database, three main steps are followed.

Finding Projects
GitHub is queried to find all the open source projects which contain at least one docker compose file of type docker-compose*.yaml. 473,131 repositories are found and saved as a repository list. This list is shuffled and divided into six lists to be distributed over worker machines.

Project Execution and Obtaining Traffic Traces
In DYNAMO setup, six worker machines are used for capturing traffic traces from repositories. They all have 4 cores, 16 GB RAM. Each machine has a repository list that it has to go through and capture the traffic from.

Once the lists are distributed, user has to manually start the capture process in each worker machine. Capturing is done according to the repository list order. During capture process, the repository is downloaded from GitHub and its docker compose files are found. Each docker compose file corresponds to an entrypoint in the project and it is run by using docker compose tool. Its traffic is captured by checking the network traces between its network interfaces and saved as a pcap file. The common output of the capture process is the following:

1. pcap folder which contains all the pcaps of captured repositories
2. info.csv file which contains metadata belonging to the capture process such as repository name, commit version, success of capturing traffic, pcap file name if any traffic is captured, duration of each step in capture process etc.
3. capture.log which contains all logging information from capture process of processed repositories

Statistics and Behavior Analysis of Traffic Traces
Since all worker machines work individually and produce their individual dataset, users have to manually merge all the datasets before doing statistics and analysis. Datasets from each worker machine is mirrored to another machine where users can merge them by combining their info.csv file and pcap folder. Users have to filter the duplicate repositories and check duplicates in pcap file paths so that no information is lost by overriding the data.

After the merging step, the analysis is done on the entire dataset and consists of two steps. The first step is to find successful entrypoints and extract basic statistics about their captured traffic traces such as tcp packet count, udp packet count etc. An entrypoint is considered successful if it produces any pcap in the capture process. Second step of analysis is done only on entrypoints whose pcaps contain IPV4 traffic. There are entrypoints which are run successfully but produced only trivial traffic. For example, their pcap contained only few "hello" or "aliveness" messages. The main goal of the second step is to extract features from pcaps such as categories of the traffic, udp/tcp flows etc. The output of the second step is an input to the Big Traffic Database.
Chapter 3

Design

In this chapter, we introduce the novelties we bring to current DYNAMO pipeline and architecture. Two main goals of the thesis are to implement the centralized architecture in DYNAMO and make the system to be ready to support future ideas without rerunning the entire process again.

3.1 Motivations

One of the downsides of DYNAMO is that there is no centrality. All worker machines work independently and there is no control mechanism over their states. In case a worker machine is stuck and no longer captures the traffic traces from repositories as expected, it is not possible to realize it without user manually logging in each machine and checking its process. Since each worker machine has different subsets of repositories to process, if one worker machine is stuck or misbehaving, it can’t execute the following repositories in its list until the worker machine is cleaned up. Therefore, high level of manual maintenance is needed on the worker machines. Moreover, when worker machines capture the traffic traces, the data is saved on the local machine. To do a traffic statistics and behavior analysis over the entire dataset, user has to merge the datasets from each machine manually and then run the analysis.

To reduce the need to manual maintenance of worker machines, we decided to automatize DYNAMO’s offline phase which consists of project execution and analysis of captured traffic. With the centralized design, we aim to have a robust system that doesn’t require much user input with facilitated monitoring. Moreover, centralized design allows us to scale the project. We can modify the architecture without effecting the rest of the system: add more worker machines, remove one worker machine, changing the configurations on one machine for experiments etc. On the top of that, we implement many features on the capture process to keep the process smooth and efficient and to prepare the project for future implementations.

3.2 Centralized Architecture

In the previous setup, we had six worker machines which were responsible of capturing the traffic in given repositories. Within the centralized architecture, we keep these worker machines and we introduce a central controller which is responsible of distributing repositories to worker machines and post processing their data. Our implementation is based on server-client architecture where central controller behaves as a server and worker machines behave as clients. In this architecture, server and clients have distinct tasks where server gives commands to clients to execute certain tasks.
As seen in Figure 3.1, the central controller has a repository list as an input (1). This list is generated by finding repositories which contain docker compose files on Github. Once central controller and worker machines are initialized, the communication between them starts. Given a repository list, central controller distributes the repositories to worker machines one by one. It waits in idle until one worker machine finishes capture process for the given repository and returns the necessary data back to the central controller. In figure 3.1, the central controller sends the repository "ganchito55/docker-taller-v" to the worker machine 1 (2). Once the worker machine finishes capturing traffic in this repository, it compresses all the output data from capture process and sends it back to the central controller (3). Central controller decompresses them and saves in the dataset location. Afterwards, it directly runs the post processing on the received data (4) and save the produced output of post processing in the same dataset location (5). When post processing is done, central controller sends the next item in the repository list to the worker machine which is waiting in idle. This process (2-5) repeats until either worker machine or central controller is shut down or there is no repository left in the repository list which is not processed yet.

On the other hand, worker machines are only responsible for capturing the traffic traces and extracting metadata in given repository. As seen in figure 3.1, worker machine starts capture process when it receives a repository name. It starts capturing traffic on the given repository (1) which triggers capturing traffic traces from each entrypoint in the repository (2). When each entrypoint is run, its results and corresponding files are saved temporarily on the worker machine (3). Once capture process is finished for all entrypoints, worker machine compresses all the outputs and transfers to the central controller. It waits in idle until another repository name is received.
One of the major design changes is that we don’t store anything anymore in worker machines. They are only responsible for capturing the traffic on given repositories and transferring the data back to the central controller.

3.3 Improvements on the Capture Process

On top of centralized design, we propose improvements on the capture process to make the dataset more structured, clean and open to future ideas. We have implemented changes in structural level and data extraction level while increasing the performance and reliability of the system.

3.3.1 Subfolder Structure in Dataset

In DYNAMO’s setup, we processed many repositories sequentially and saved their results in common files and directories. For instance, we had one common info.csv file in each worker machine to save metadata of capture process and one pcap folder for all captured pcaps. In case we wanted to get data about one repository, we needed to find its entry on metadata file and extract its pcap file’s path from the metadata. This structure wasn’t easy to use for manual inspection or debugging.

With the new centralized architecture, each worker machine captures traffic traces from one repository at the time and returns its data. In addition, we propose having a subfolder structure in repository level. Considering that capturing the traffic for a repository means capturing the traffic for each entrypoint in a repository, it is meaningful to save results and corresponding files of each entrypoint in a different folder. Below, we display how the repository ”ganchito55/docker-taller-v2” is structured in the dataset.

```
ganchito55
  └── docker-taller-v2
      ├── README.md
      ├── capture.log
      └── 0_entrypoint
          ├── analyze.log
          ├── docker-compose.yml
          │    └── docker_info.csv
          │        └── info.csv
          │                └── metadata.csv
          │                        └── pcap
          │                                └── README.md
          └── 1_entrypoint
              ├── analyze.log
              │    └── docker-compose.yml
              │        └── docker_info.csv
              │                        └── info.csv
              │                                └── metadata.csv
              │                                               └── pcap
              │                                                   └── 81962c26-17f5-41c4-aac5-7324142d5306.pcap.gz
              │                                                       ├── analytics
              │                                                           └── 81962c26-17f5-41c4-aac5-7324142d5306.tcp_flows.csv
              └── README.md
```
3.3. IMPROVEMENTS ON THE CAPTURE PROCESS

With new subfolder structure, each subfolder will correspond to an entrypoint in the processed repository and contains the following data after capture process:

1. info.csv which has metadata about processed repository, corresponding entrypoint and capture process of that entrypoint
2. docker.info.csv which has metadata extracted from the docker compose file of that entrypoint
3. docker compose file corresponding to that entrypoint
4. closest README file to the docker compose file in project repository
5. pcap if entrypoint was successfully run and produced any traffic

After postprocessing, subfolder content is expanded to include following data:

1. analyze.log which contains logging information about post processing each entrypoint
2. metadata.csv which has traffic analysis about captured traffic in this entrypoint
3. udp flows and tcp flows if any flow is found in pcap during postprocessing

In the example above, we see that only the second entrypoint is successfully run and pcap is produced. Moreover, its captured traffic traces include tcp flows which is saved under pcap/analytics directory.

3.3.2 Dataset Content Expansion

While the dataset is shaped to have a clear structure, we also started extracting more metadata and files from repositories during capture process. These data can be used in future to better understand the repository behavior and to match its traffic traces with applications.

Extract Docker Compose File

To be able to understand what a container does, it is a good idea to look at its docker compose file. This file defines services, networks, volumes and more configurations of the container which help us to understand container’s application. After analyzing docker compose files, it is easier to understand its captured traffic traces and define their roles in the applications. Therefore, we extract the docker compose file for each entrypoint and its path is added to info.csv file to facilitate locating the file.

Extract README File

To be able to understand what the container’s purpose is and how it is supposed to run, it is a good idea to look at its README file. If exists and well defined, this file explains the application in high level and contains information about how to run the docker compose file: what are the necessary or optional parameters, flags and other configurations. In DYNAMO, it is observed that one of the main reasons that repositories failed to execute is that default command, docker compose up, is not enough to run containers. Therefore a future improvement can be to parse already extracted README files to find the correct command to run the container.

A repository may contain many README files. Generally, the most specific information about an application is found in the same directory of the application files. In our case, the README
3.3. IMPROVEMENTS ON THE CAPTURE PROCESS

which is in the same directory of docker compose file is the most insightful one. If there is no README file exists in the same directory as the docker compose file, README in the parent directory may contain the most information about the docker application. We need to go one parental level up recursively until we find a README for this entrypoint or we reach the top level directory of the project. For each entrypoint, we extract the README file closest to its docker compose file and its path is added to info.csv file to facilitate locating the file.

Extract Docker Metadata

When a docker compose file is run, its containers are built and we capture the traffic between network interfaces of these containers. Therefore extracting features about docker containers can help us to better understand how container behaves and map captured traffic traces to the network interfaces. Docker metadata is promising to be used in future to further analyze the containers and develop features. For instance, we can keep track of success rate of containers. If we know that a particular container to be failed before, we don’t have to run it again if we come across to this container in another project. That improvement can fasten the capture process while avoiding wasting resources on repetitive containers. We can also do a statistical analysis on container’s use frequency among repositories.

We use docker service and docker daemon to extract following metadata about docker containers:

1. Network interfaces is the list of interfaces which are built while docker containers are up and running. We capture the traffic between these interfaces, therefore knowing the interfaces are important to understand if containers communicate with each other or with the internet directly.

2. Docker command is the command used to run the docker compose file. In this thesis, we run all projects with command ”docker compose up”. A future improvement is to parse documentations and project information to find the actual command to run the file.

3. Docker container ids, names, images, labels, status and sizes are directly extracted from docker service. Storing ids and images are useful to analyze container behavior such as if we run one image or container frequently or if success rate of the projects change even if they use same images. Many docker containers are using official images of well known services to build their applications. If we already know how these images behave, we can easily explain the captured traffic.

4. Docker container network settings is a configuration field in docker compose file. We use docker daemon to extract network settings and if it is defined, it gives many information about the network configurations of containers such as global IP addresses, port numbers etc.

3.3.3 Stability and Maintenance of Worker Machines

During the thesis, we observe that the most frequent reason that worker machines get stuck is failure of docker services. With docker services failing, all the sequential projects get stuck or fail too. To fix this problem and assure the system stability, a docker service cleanup step is added to the capture process. Before running each entrypoint, we restart docker service and clean up previous docker data so that system stays more reliable. Since we integrated docker cleanup step, it is not observed that a worker machine is stuck for this reason. Restarting docker services are shown to be efficient while not delaying the capture process with a significant amount.
To track capture process in worker machines, we developed a daily updating system. Everyday, maintainers of the projects get notified about status in each worker machine: how many repositories processed that day and their success rate. We also inform users if there is no output or no successful output from any worker machine since it may indicate that worker machine is stuck and needs to be cleaned up. This updating system is helpful to keep maintenance on worker machines with little effort. To be able to track which worker machine processed the repository from central controller, we expanded info.csv to contain worker machine number.
Chapter 4

Evaluation

Within the scope of the thesis, we are interested in understanding our dataset. We first do a manual inspection on selected repositories and their pcaps to see if captured traffic traces are good representative of container’s application behavior. Later, we do a statistical analysis on the dataset to see general behavior of repositories and try to conclude our assumptions.

4.1 Discussions and Observations on Selected Repositories

To match the traffic with applications, we conduct a manual inspection on selected repositories. By inspection, we aim to test our hypothesis that collected traffic traces show the behavior of the container. Since they are no tools or automatized system which can help us to compare the container’s purpose and its produced traffic, this step had to be done manually. The inspection is done on few repositories which had entrypoints that produced pcaps. Since we processed 293131 repositories, it wasn’t possible to do a manual inspection on all of them.

To compare the traffic with the container application, we first need to understand what container aimed to do. Understanding its goal is not possible from the dataset, thus we need to read its documentation and inspect code repository. Docker compose files may contain names or images of well known services which explain the container’s application as well as its produced traffic. Moreover, README file in that repository contains the information about the container and how it works. Inspecting the code, the variable names and the file names also give us ideas. We did this inspection with detail on few repositories and try the match the traffic with applications.

We observe that reading the documentation and inspecting the repository is very useful understand the container’s application. Some docker compose files contain official images of well-known services such as postgresql, nginx, syslog, kafka and many more. If an application uses a well-known service whose network traffic is known, it is straightforward to compare the traffic that we captured from the container. We give two examples to better describe our inspection process and to support our claim.

1. We inspect the docker compose file from repository manishnepali/DevV. It is using the image "postgres:latest" which the official image of PostgreSQL database. Moreover, its environment variables defines the configurations to connect to the database such as postgres password, username and db. When we read its README file, we learn that it is an application to pluralize the words. Our initial intuition is that this application uses PostgreSQL database to find the plural version of the word and we expect a traffic towards to this database. When we look at the traffic traces that are obtained from running manishnepali/DevV, we see that it
4.2. STATISTICAL ANALYSIS ON BIG TRAFFIC DATASET

uses destination port number 5432 which is widely used to reach the PostgreSQL database[6]. Our intuition is confirmed by checking the destination ports of traffic traces and we could map the traffic to the application.

2. The docker compose file from adrianvillanueva997/Dockerfiles/VPS/gamingservers/arma3 run successfully and we captured the traffic traces. From the first glance at the repository on Github, we see that file exists in directory gamingservers. Knowing that arma3 is a video game, it seems like that application is related with gaming. There is no particular documentation or README in this folder, so we inspect its docker compose file. Container builds arma3 by using its docker image and in its environment field Steam username and password is defined. Steam is an online video game streaming service, owned by Valve Cooperation [7]. From manual inspection, it seems that it is an application which builds up arma3 game and stream on Steam. Once pcaps are analyzed, our hypothesis holds. We observe that there are traffic traces which uses destination ports belonging to Steam service [8] and destination IP address belonging to Valve Cooperation. Thus again, we could see that captured traffic reflects application behavior.

Repositories with many entrypoints are analyzed to understand the factors why some entrypoints succeeding while others failing. We observed that there are many repositories with more than one entrypoints but only one of these entrypoints succeeded. Doing manual inspection helped us to realize one reason of this situation. Many repositories contain versions or modified duplicates of docker compose files, including very basic docker compose files. Some versions of compose files are basically empty, only version number is defined. They are built and run successfully however its captured traffic doesn’t involve any interesting traffic. It contains some aliveness messages but it doesn’t produce any traffic itself.

Using well known images is an important factor in the success rate of containers. When a container is built only on an official image of well known service, it can be run and traffic is captured successfully. Official images are built in a robust and standalone way and is designed to run efficiently without extra inputs. In manual inspection, we observe many successful entrypoints use well known images. However, when compose files combine official images with their own implementations, they often fail and no traffic is captured.

4.2 Statistical Analysis on Big Traffic Dataset

We are interested in understanding our preliminary results and dataset to improve efficiency of the capture process. This helps us to define future steps of the project to create a useful database where live traffic can be generated from. We first do a statistical analysis on results obtained from dataset created on 30-09-2022 and used in DYNAMO [1]. In this dataset, we captured 293131 repositories and 1067318 entrypoints.

We compute the distribution of entrypoints per repository. From manual inspection, we see that most repositories have one or two entrypoints. Our assumption is proved with the distribution in figure 4.1. We observe that we processed 388 entrypoints which have more than 100 entrypoints. Capturing more than 100 entrypoints within the same project may be quite heavy on the system which requires lots of resources and potentially end in failing or sticking the worker machine. To not waste resources on worker machines and keep our system robust, we changed our setup to process at most 100 entrypoints per repository.
4.2. STATISTICAL ANALYSIS ON BIG TRAFFIC DATASET

Majority of the entrypoints are observed to fail in preliminary data and our manual inspection. To understand the general behavior of repositories, we find the distribution of success rate per repository where success rate is defined in equation 4.1. In figure 4.2, we see that majority of repositories are failed completely. There might be different reasons for a container to fail such as general docker command was not enough to run the container, there is a bug in the container code, there is a bug in our code etc. As a future improvement, we would investigate better the reasons of failures and parse documentation to get the exact docker command.

\[
\text{successRate} = \frac{\text{succeededEntrypoints}}{\text{succeededEntrypoints} + \text{failedEntrypoints}}
\]  \hspace{1cm} (4.1)

An output of manual inspection was the existence of repositories with many entrypoints but only one succeeded. As seen in table 4.1, this behavior is observed often enough and needs further investigation. We should do an analysis to understand how often we have the basic compose files and if their existence impacts our database. A future improvement in the project can be filtering docker compose files and not run the ones which don’t contain any implementations or services.

<table>
<thead>
<tr>
<th>Total number of repositories</th>
<th>293121</th>
</tr>
</thead>
<tbody>
<tr>
<td>With more than one entrypoint</td>
<td>124480</td>
</tr>
<tr>
<td>With more than one entrypoints but only one entrypoint succeeded</td>
<td>9677</td>
</tr>
</tbody>
</table>

Table 4.1: Repositories with multiple entrypoints
In the thesis, Docker Hub rate limiting is observed to be the biggest bottleneck of the system. We question if using a personal vs. pro account can have an impact on success rate of the projects as well. We conduct an experiment where we tried to capture the traffic in a set of 1040 entrypoints. Our initial setup had two personal Docker Hub account where each account is used by three worker machines. Later setup had three pro Docker Hub account where each account is shared by two worker machines. We run same entrypoints with both setups (might be different commit versions of repositories) and observe if running them with pro account has any effect on the success rate of the entrypoint. As we see in 4.3, the status of the most entrypoints remained same. There are some status changes observed but our data is very limited to make assumptions about ground truth without analyzing other factors which may have an effect on the running process. Since we run both setups at different times, it is possible that containers, codes or packages got updated which may impact their status. Therefore, we conclude that using pro Docker Hub accounts don’t have any observable positive influence on the success rate on this dataset.
Figure 4.3: Entrypoints’ status change when run with pro Docker Hub account
Chapter 5

Outlook and Summary

In this thesis, we worked on the continuation of DYNAMO where was no central system. All parts of the project were run separately. Users were responsible of combining each part, forming the dataset, and maintaining the worker machines. With the thesis, we managed to design a centralized system which works in a robust way without requiring too much user input or manual work. Capturing traffic traces from repositories, extracting traffic features and obtaining metadata is done automatically. The dataset is updated every time a repository is processed. With the central architecture, the project can be easily scalable. We can modify the architecture without effecting the rest of the system. On the top of designing the central system, we also changed the structure of the dataset to make it more accessible and easy to use. Furthermore, we expanded the data that we collect during the capture process while extracting more metadata and related files from repositories. We also improved the capturing process by making it more reliable and easy to monitor.

As mentioned through the report, there are many future improvements on the project. To mention a few, we can parse repository documentation to find docker compose run command, extract more docker metadata such as environmental variables, ports, tags and well known images. We should do further analysis on our dataset to understand the reasons of failures in entrypoints and investigate the behavior of repositories with many entrypoints. Our project is dynamic and ready to support other future ideas and improvements without rerunning the entire process again.
Bibliography


