Instituto Superior Técnico, Universidade de Lisboa

Large-Scale Systems Engineering

Laboratory Guide – Part I
Deployment, Monitoring and Basic System Analysis

Goals

The ability to quickly deploy, measure and reason about a system's behaviour is fundamental. The goal of this laboratory guide is to get familiarized with some of the tools available for this, namely:

- Docker, a container technology that facilitates the deployment and management of large-scale applications
- Gnuplot, a programmable plot drawing tool that can be used to quickly plot metrics of interest about a target system
- dstat – a command line monitoring tool for Linux systems

We will use all these tools to reason about the behaviour of a simple application.

Preliminary notes

- Sometimes when copying the commands in this guide, characters such as dash (-), backtick (`) or quotes (") among others get automatically converted into other characters by the terminal application or operating system. If a command fails with a syntax error, double-check that the proper character is being used.
- Some Linux distributions ship with a non-official Docker version with slightly different behaviour. Whenever possible use the official Docker installation. Instructions available at https://docs.docker.com/engine/install/
- Due to the fast-paced evolution of Docker, older versions might not support all functionalities we use in this guide, or might use different syntax for some commands. Check if you Docker installation is up to date.

1. Docker

Docker allows to package applications in containers that are easy to build, deploy and distribute. Containers are a lightweight virtualization technology that is more resource efficient than regular virtual machines (as resources like the host kernel are shared among all containers) but also provide less isolation and security guarantees.

Docker applications are packed into images that can be easily distributed and deployed. It is possible to run custom private images and to run publicly available images provided by third parties.

Before proceeding make sure Docker is installed on your machine. For instructions on how to do this check the official Docker documentation at https://docs.docker.com/install/.

In the examples below we will use the notation

```
(host)$ <command>
```

to indicate a command should be run on the host (your machine) and

```
(container)$ <command>
```

to indicate a command should be run on the container named *container*.

To list the images available on the local system run:

```
(host)$ docker images
```

To run an example image:

```
(host)$ docker run -it --name test1 ubuntu:23.04
```

this will download the version *23.04* of the publicly available *ubuntu* image, name it *test1* and open a shell inside the container in interactive mode (*-it*). The container is an independent entity with its own process space and network stack. To check the processes running inside the container:

```
(test1)$ ps -a
```

Let's check the IP address of the container. We need to install the iproute2 suite first.

```
(test1)$ apt-get update
(test1)$ apt-get install -y iproute2
(test1)$ ip -4 addr
```

Open another shell and check the containers currently running:

```
(host)$ docker ps
```

Each container will have an ID and a name. If the name is not specified by the user, a random name will be generated.

To obtain detailed information about the *test1* container just created:

```
(host)$ docker inspect test1
```

The information is very detailed and useful for debugging purposes. For instance, to obtain the IP address assigned to the container from the host:

```
(host)$ docker inspect test1 | grep IPAddress
```

The container will run until the command being executed (the default shell in this case) finishes. To terminate the container:

```
(test1)$ exit
```

this will terminate the running shell and, consequently, terminate the container.

It is also possible to execute custom commands in a container by passing the commands as a parameter. For instance

```
(host)$ docker run ubuntu:22.04 ps -a
```

This will create a new Ubuntu container and run the “ps -a” command inside it. Because the command executes and terminates, the container will terminate immediately. Now try:

```
(host)$ docker run ubuntu:22.04 ip -4 addr
```

The first command runs successfully but the second returns an error. Why?

To check all containers in the system, including terminated ones:

```
(host)$ docker ps -a
```

and to remove unwanted containers, for instance the *test1* container created before

```
(host)$ docker rm test1
```

Now let's verify container networking. Start a new container:

```
(host)$ docker run -it --name ncServer ubuntu:23.04
```

and install netcat a networking toolkit. Run a netcat server inside the container

```
(ncServer)$ apt-get update
(ncServer)$ apt-get install -y netcat-traditional
(ncServer)$ nc -l -p 4000
```

This will run a *netcat* server that prints to the standard output all the input received over the network. Obtain the IP address of the *ncServer* container using one of the methods above, and in another shell start the client container:

```
(host)$ docker run -it --name ncClient ubuntu:23.04
```

In the *ncClient* container install netcat and connect to the *ncServer* netcat service.

```
(ncClient)$ apt-get update
(ncClient)$ apt-get install -y netcat-traditional
(ncClient)$ nc <ncServer IP> 4000
```

Now type some words (and press enter) on the *ncClient*. These should appear in the *ncServer* output.

The connection is bi-directional so typing on the *ncServer* container will print the output in the *ncClient* container as well.

Terminate both containers.

As you saw, each time a container is launched is starts from a fresh state. Let's create a Docker image that automatically starts a netcat server on port 4000 by default. This can be achieved by extending an existing image. To do so, create a new directory called mycontainer and change to it

```
(host)$ mkdir mycontainer
(host)$ cd mycontainer
```

Inside the *mycontainer* directory create a new file named *entrypoint.sh* with the following contents (note the backticks on line 3):

```
#!/bin/sh
echo "Hello world"
echo my ip is `ip -4 addr`
nc -l -p 4000
```

This will be the script executed when the container starts.

The next step is to extend the original ubuntu image to include our custom script. In the same directory, create a file named *Dockerfile* with the following content:

```
#extend the original ubuntu image
FROM ubuntu:22.04
#copy the starting script to the container image
COPY entrypoint.sh /entrypoint.sh
#install ping, iproute and netcat
RUN apt-get update && apt-get install -y iputils-ping iproute2 netcat-traditional
#run the custom script using the “sh” shell
CMD ["sh", "/entrypoint.sh"]
```

Now build the image with

```
(host)$ docker build -t mycontainer:latest .
```

The image will now be available in the local images. Check it with:

```
(host)$ docker images
```

Now run the newly created image:

```
(host)$ docker run -it mycontainer
```

You should see the initial hello world message, together with the IP address of the newly created container.

Next, we will start a new client container using the newly created image and check that we can connect to the server.

Sometimes, it is necessary to install new packages when extending an existing container, or run some preliminary configuration scripts. This can be achieved with the RUN instruction on the Dockerfile as in the example above. Note that the default command (given by the CMD instruction in the Dockerfile) can be overridden by

providing an alternative command. Let's try it by starting a new instance of the *mycontainer* image previously but instead of running the default command (starting the netcat server) we start a shell:

```
(host)$ docker run -it mycontainer /bin/bash
(container)$ nc <ncServer IP> 4000
```

Terminate both containers.

2. Gnuplot

Gnuplot is a very powerful plotting tool that can plot 2D and 3D graphs using raw data or functions. It is command-line oriented and hence easy to script and automate.

On Linux based systems, Gnuplot is available in the regular package management tools. For Windows/MacOS there are precompiled binaries that can be easily installed.

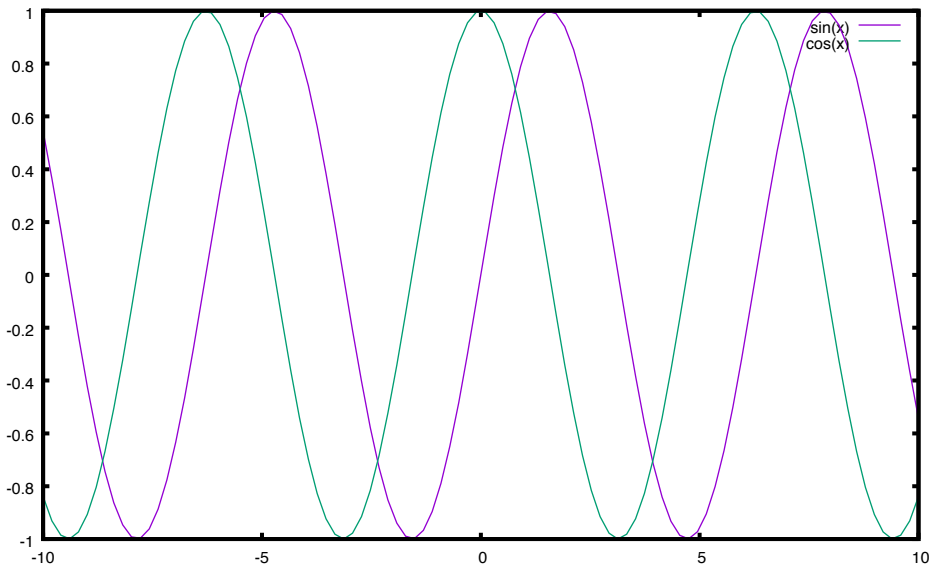
Start by creating a simple file, named *functions.gp* with:

```
#this will generate the plot in PDF format
set terminal pdf
#the plot will be saved to file functions.pdf
set output 'functions.pdf'
#plot the sin and cos functions
plot cos(x) ,\
     sin(x)
```

To generate the PDF:

```
$ gnuplot functions.gp
```

This will generate a file named functions.pdf. Check the resulting PDF which should look like this



Gnuplot usually assumes reasonable defaults for the plot parameters.

Let's say we wish only to see the positive X range of the plot, from zero to 10.

This can be achieved by configuring the xrange parameter (and yrange for the Y axis):

```
set terminal pdf
set output 'functions.pdf'
set xrange[0:10]
plot cos(x) ,\
     sin(x)
```

Redraw the plot, by running Gnuplot, and check the result.

Plotting functions is useful, for instance when assessing the observed behaviour of a system with respect to the behaviour predicted by a model.

Gnuplot can also plot data points from a data file.

Suppose you collected the throughput of some application with an increasing number of clients. Create a file named *throughput.dat* with the following content:

```
#Clients Throughput
1    500
2    900
5   2250
10 4500
20 9000
```

To plot that data in Gnuplot, created a file named *throughput.gp* with the following:

```
set terminal pdf
set output 'throughput.pdf'
#use the first and second columns from the file throughput.dat
plot 'throughput.dat' using 1:2 title "throughput"
```

This will generate a plot of the data points. Most likely, the Y axis will start at 500 rather than zero. Adjust the xrange accordingly and redraw the plot.

It is also fundamental to have the proper labels in both axis, and a plot title. This can be achieved by adding the following parameters (which should appear before the *plot* call):

```
set xlabel 'Clients'
set ylabel 'Throughput (op/s)'
set title 'Application Throughput'
```

To observe the data trends sometimes it is helpful to draw a line connecting all the data points. This can be achieved with the linespoints style, as follows:

```
plot 'throughput.dat' using 1:2 title "throughput" with linespoints
```

Now suppose the data needs to be represented at thousands of operations per second. This can be achieved by manually editing the raw data to present the throughput in those units but it is more convenient to let Gnuplot do this, as follows:

```
plot 'throughput.dat' using ($1):($2/1000) title "throughput" with linespoints
```

Note that we now use references to the data columns, with the $ symbol.

The Y label is now incorrect as the data is presented in thousands of operations per second. Update it accordingly.

Often, it is more useful to represent data with a bar plot. Let’s say we have three possible configurations for a given system with different costs in file configs.dat

```
#configuration name cost
0 "config A" 100
1 "config B" 450
2 "config C" 75
```

To generate the corresponding bar plot, use the following Gnuplot file:

```
set terminal pdf
set output 'configs.pdf'
set xlabel 'Configuration'
set ylabel 'Cost (eur)'
set title 'Configuration cost'
set boxwidth 0.5
```

```
set style fill solid
set style line 1 lc rgb "blue"
plot 'configs.dat' using 1:3:xtic(2) with boxes linestyle 1
```

In the plot command we use the first column as the X axis, the third column as the Y axis and the second column as the labels in the X axis.

Every plotting aspect in Gnuplot is highly configurable, for further information check the references below.

3. Monitoring

To understand system behaviour, it is fundamental to monitor several key system metrics such as CPU, memory, disk and network input/output, among others.

dstat[*] is a powerful tool that can be used to monitor many performance aspects of a Linux system.

Some of the most common options when using dstat include

```
-c cpu
-d disk
-m memory
-n network
```

For instance, to measure CPU and memory usage every 2 seconds, and take 30 samples:

```
dstat –cm 2 30
```

For containers, and when access to the host is available, it is also possible to use docker stats which show the live stats of all containers.

```
(host)$ docker stats
```

4. Putting it all together

The goal is to assess the behaviour of the popular *nginx* server when subject to increasing load. To this end we will use the publicly available nginx:latest Docker image.

- Start a container with the nginx:latest image
- Obtain its IP address
- Test the connectivity to the nginx server. You can use the curl tool to access

[*] **Note**: dstat can be installed on ubuntu:22.04 with `apt-get update && apt-get install dstat`

the welcome page of nginx as follow:

  - curl http://<SERVER IP>:80
  - If the server started correctly, you should see the html of nginx welcome page.

- To benchmark the system, we need to put load in it. To this end, we will use siege, an http load injection tool. To prevent the siege from overloading the physical host, we can limit its CPU usage:

```
docker run --cpus=0.01 jcabillot/siege --concurrent=1 --delay=1 --time=1m http://<SERVER_IP>:80
```

  - This will start siege, with one concurrent client issuing requests with a random delay between zero and one seconds. The benchmark will run for a total of 1 minute. The container is limited to use 1% of a physical core.
  - After finishing, the benchmark will report several metrics, including the throughput. Measure and plot, the throughput for 1,2,5,10 and 20 clients.
  - Are the results obtained expected? How can they be explained? Is nginx under heavy load? Note that the throughput can be expressed by several different metrics.
  - Validate your previous hypothesis by using dstat on the nginx container to assess the container load. What is the limiting factor?
  - How could the results be improved?

Bibliography

Docker documentation - https://docs.docker.com/get-started/
Gnuplot in Action: understanding data with graphs. Philipp K. Janert. Manning
Gnuplot tutorial - http://physics.clarku.edu/sip/tutorials/gnuplot/
dstat man page - https://linux.die.net/man/1/dstat