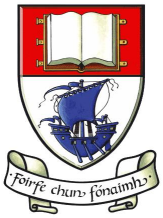

Waterford Institute of Technology

Cloud Infrastructures

Collecting Detailed Metrics from Docker Swarm Hosts

Stephen Coady

20064122

BSc in Applied Computing

March 24, 2017

Contents

1 Introduction

The aim of this research paper is to demonstrate how, using several different technologies, a swarm host can be dynamically polled and the collected metrics displayed graphically. It will show this method of data collection can be turned into a template so that it can be re-used for each new host which is added to the swarm.

It will also show how this process can automated so that no action is required from the administrator each time a new host is added to the swarm or a new service is created.

This paper will also examine the options available for real-time or live feedback on these hosts, for example an alternative to Amazon Web Service's CloudWatch. It will then compare these tools and processes to those available natively by using Docker. It will evaluate the strengths and weaknesses of both.

2 Problem Statement

As of July 2016 Docker usage had exploded in popularity, with some sources stating the container based technology's usage had risen by over 3000 percent in the two years since it left beta release (Arijs, 2016). With this in mind, it is natural to conclude that the amount of 'Dockerised' applications running in production has also grown at a fast rate.

Since an application running in production is assumed to be serving real customers or users downtime should be minimised and the performance of the application should be maximised (i.e. response time, latency, caching). The first part of this is taken care of by Docker Swarm, which provides scaling of applications across multiple hosts. The second, however, is not an easy problem to solve. Since Docker containers are supposed to be ephemeral it is difficult to monitor them as their short lifespan is not conducive to gathering longterm meaningful statistics (Young, 2017). This problem is only exacerbated by the fact that Docker Swarm may create multiple versions of the same container running across several different hosts.

While the Docker remote API does provide some metrics from containers natively it is less than ideal to need to keep track of each individual host and then to poll these hosts for metrics relating to each container on the host - especially since we may not know which service or indeed even how many services are running on each of our hosts. If we are then running our application in an elastic computing fashion such as Amazon's EC2 then these hosts may also be ephemeral and so the monitoring problem quickly becomes unmanageable.

We will now look at some of the technologies involved in the problem and the technologies we are proposing to use to solve the problem.

3 Technology Background

Since the aim of this paper is to implement a monitoring system which is not tied to any one vendor such as Amazon Web Services or Microsoft Azure it will aim to use only open source and community driven software to deliver this solution. This will mean that all findings should be easily reproducible and applicable to any cloud provider.

3.1 Containers

A container uses Linux abstraction techniques such as namespaces and cgroups to isolate and 'contain' a software process from those running on the same host. It does this limit resources available to that process while also making it much more manageable for the user. The container itself does not know that it is a container but instead is presented with a local resource and made to believe it has access to a global resource (Kerrisk, 2013). Containers and their workings are outside the scope of this paper, however it is important to understand that they are simply a process running on a host.

3.2 Docker

Docker is a piece of software which manages the orchestration of containers. It allows the user to tell Docker what they would like Docker to do and it leaves the underlying Linux translations to the Docker engine (Docker, 2016*b*). There are many alternatives such as Kubernetes and Amazon's Container Service, both of which can also manage Docker containers if the user wishes.

Docker receives its instructions from the user in the form of a 'Dockerfile' which is essentially a set of instructions specifying details such as the base image to use and what commands should be run on the container once it starts. This image is then built and can be used as a template for many containers to run from. It is this property which results in a single Docker host having multiple containers all based on the same image - much in the same way a Docker swarm host may contain many containers running a single service as previously discussed in Section 2.

3.3 Docker Swarm

Docker Swarm is a mode of Docker whereby a master/slave relationship is enforced across multiple internet-connected hosts with the express intent that the applications started on these hosts would be spread across all hosts in the *swarm* (Docker, 2016*a*). For example, in a swarm of say 5 hosts, the user could issue a command on the manager node to start a service which is made up of 10

containers. These 10 containers would then (more than likely) be spread across all 5 hosts with each running 2 containers. This means that the application now has 5 different routes by which it can be reached with a fault tolerance of an extra container on each host. While this example is trivial and not very practical it is intended to give a basic understanding of Docker swarm mode.

3.4 Grafana

Grafana is a graphical dashboard which can be used to display information from many different types of database. It allows very fine-grained control over the metrics displayed and also how they are displayed. It is a fully-functional web application which supports user login and different views depending on the logged in user (Grafana, 2017). It is not core to this project and there are many alternatives, however it was decided that Grafana was the best option as it has a large online community and support for a large number of third party software packages.

3.5 Prometheus

Prometheus is an open source and community driven monitoring solution which provides a multi-dimensional data model under which any metric can be stored efficiently. It also provides its own query language which can be used to extract information from these metrics (Prometheus, 2017).

Some of the major benefits of Prometheus over other metric storage solutions are:

- A package called 'Node Exporter' which is built by the same community has native compatibility with Prometheus and can be used to collect metrics from the host it is running on. This is extremely useful when monitoring a multi-host system such as a Docker Swarm.
- It has native support for Grafana.
- It also has native support for tools such as cAdvisor, which we will discuss further in Section 3.6.
- It has an active and large community.

3.6 cAdvisor

Container Advisor (cAdvisor) is a standalone application which provides detailed feedback about all containers running on a host. The metrics it provides include performance characteristics such as CPU usage, memory usage and network activity (Google, 2017).

cAdvisor has native support for many different types of containers, including Docker, so it is the ideal choice for this paper.

While it does provide a web app which can graphically display the information its real power lies in the fact that it also exposes a remote API which can be scraped for the metrics. We will take advantage of this fact in Section 4.

3.7 Infrastructure Overview

Now that we have seen each technology individually it is not difficult to see how each will fit together to provide a cohesive and modular application which will monitor both our Docker host and the containers which it is running. We can see a diagram giving a visual guide in Figure 1 below.

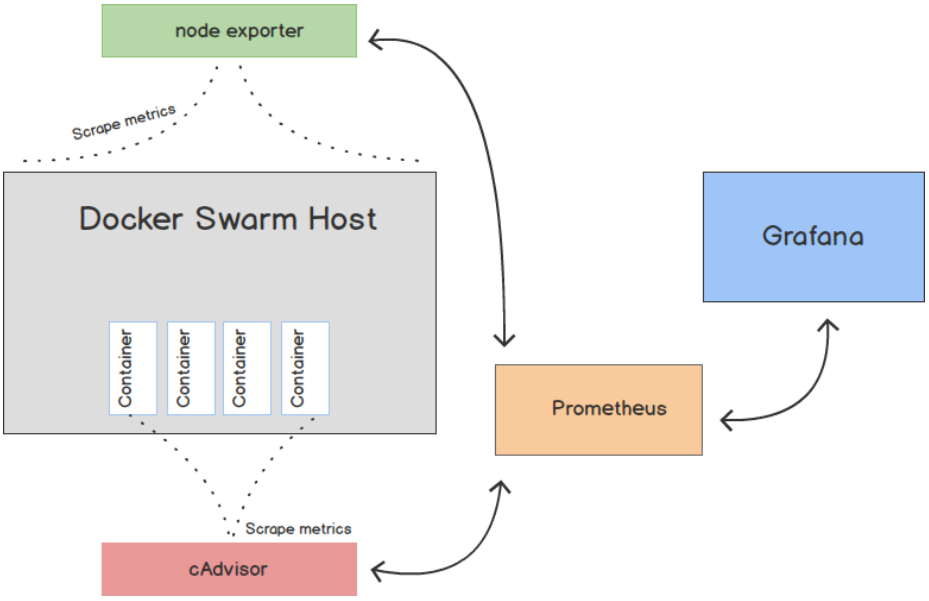

Figure 1: *Infrastructure Overview*

- The node exporter will read metrics based on the Docker Swarm host
- The cAdvisor component will read metrics based on the containers running on the host
- Both cAdvisor and the node exporter will be scraped by Prometheus which will in turn store these metrics for later usage
- Grafana will then be configured to periodically poll Prometheus and display the metrics visually.

4 Practical

5 Discussion

Bibliography

Arijs, P. (2016), 'Docker usage statistics: Increased adoption by enterprises and for production use', http://www.coscale.com/blog/docker-usage-statistics-increased-adoption-by-enterprises-and-for-production-use. Accessed: 23-03-2017.

Docker (2016*a*), 'Docker Swarm', https://docs.docker.com/swarm/overview/. Accessed: 03-10-2016.

Docker (2016*b*), 'What is docker?', https://www.docker.com/what-docker. Accessed: 03-10-2016.

Google (2017), 'cAdvisor', https://github.com/google/cadvisor. Accessed: 24-03-2016.

Grafana (2017), 'Grafana', https://grafana.com/. Accessed: 24-03-2016.

Kerrisk, M. (2013), 'Namespaces in operation, part 1: namespaces overview [LWN.net]', https://lwn.net/Articles/531114/. Accessed: 03-10-2016.

Prometheus (2017), 'Prometheus', https://prometheus.io/docs/introduction/overview/. Accessed: 24-03-2016.

Young, K. (2017), 'The docker monitoring problem', https://www.datadoghq.com/blog/the-docker-monitoring-problem/. Accessed: 24-03-2017.