DockSNAP : Performance Monitoring of Docker Systems

Pratik Bhangale
Computer Science, UMBC
Email: pratikb1@umbc.edu

Vishal Rathod
Computer Science, UMBC
Email: vishalr1@umbc.edu

Mayure Pate
Computer Science, UMBC
Email: mayurp1@umbc.edu

***Abstract*—Over the last decade , the use of virtualization and container based technologies has increased dramatically. Container based virtualization and hypervisor-based virtualization are two main types of virtualization technologies popular in market. Of this two, container based docker systems are very much poplar today. Docker containers wrap a piece of software in a complete file system that contains everything needed to run: code, runtime, system tools, system libraries anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment. Due to this there is demand for efficient monitoring tools for the docker containers. But existing solution to monitor the docker containers requires entering command line commands to see the status of containers. In this paper we propose the Push based system to see the status of various containers on a web based interface. The monitoring stats gives details about (1) CPU (2) Memory usages. Furthermore, the paper also discusses and identifies what could be done to improve such systems.**

***Keywords*—Containers, Docker, Monitoring Client, Virtualization, Web Service**

I. INTRODUCTION

Virtualization technology uses a software to simulate the existence of hardware and other resources to create a virtual computer system. Doing this allows businesses to run more than one virtual system which has multiple operating systems and applications which are running on a single server. This helps with numerous things. Companies can save on large expensive resources and and it also helps with low costing Web services. The last decade has seen an impressive development in the area of virtualization technologies, which allow the partitioning of a computer system into multiple isolated virtual environments. Virtual machine migration is truly enabling attribute of virtualization technology. As this emerging technology matures, it will become even more popular. One of the important reasons big companies adopting the virtualization is server virtualization in data centres. Server virtualization is the partitioning of a physical server into smaller virtual servers which helps the companies to maximize the server resources. In server virtualization the resources of the server itself are hidden, or masked, from users, and software is used to divide the physical server into multiple virtual environments, called virtual or private servers.

Current Virtualization techniques can be classified in two major types: container-based virtualization and hypervisor-based virtualization. Container based virtualization is at the operating system level, while the hypervisor based system provides virtualization at the hardware level. Thus the container based solution is more lightweight and efficeint. One of the most popular container based system is Docker. One of the important need for such systems is to constant monitor the systems and view its stats. Existing solutions to monitor the docker containers requires entering command line commands to see the status of containers. In this paper we propose and describe the details of new push based system that allows to display the status of various containers via a Web-based interface.

The paper is structured as follows: Section 2 provides motivation for developing the docker performance monitoring tool. Section 3 gives an related work about other docker monitoring tools. Section 4 provides background and comparison of virtulization and docker based techniques, and then in further sections we elaborate the docker implementation and the followed approaches along with the security level of Docker and what could be done to raise its level of security. The paper concludes with a summary in Section 14.

II. MOTIVATION

Docker shook the DevOps world. Containers ready for cloud architecture brought production operations closer to development and helped make microservices the backbone of a more flexible, aggressive approach to build software architecture. Running various applications in Docker containers only changes how they are packaged and scheduled - not how they run. How do I monitor Docker in my production environment? this is the major concern for system administrators and DevOps . System administrators have to enter the commands on terminal to view the stats on console. Clean web interface for monitoring multiple clients is missing . This is the major motivation for developing such systems. Docker shook the DevOps world. Containers ready for cloud architecture brought production operations closer to development and helped make microservices the backbone of a more flexible, aggressive approach to build software architecture. Running various applications in Docker containers only changes how they are packaged and scheduled - not how they run. How do I monitor Docker in my production environment? this is the major concern for system administrators and DevOps . System administrators have to enter the commands on terminal to view the stats on console. Clean web interface for monitoring multiple clients is missing . This is the major motivation for developing such systems.

III. RELATED WORK

This work is inspired by the recent successes in many open source docker monitoring tools and utilities. Docker is being used in more and more production deployments. As such, the ecosystem surrounding Docker is picking up the gauntlet by creating more and more solutions for monitoring which is crucial for keeping tabs on a Dockerized environment and gaining visibility into the state and health of containers. cAdvisor (Container Advisor) provides container users an understanding of the resource usage and performance characteristics of their running containers. Daemon which is running collects then aggregates, processes, and exports information about running containers. Specifically, for each container it keeps resource isolation parameters, historical resource usage, histograms of complete historical resource usage and network statistics. This data is exported by container and machine-wide. Dynatrace offers a powerful solution for Docker monitoring by providing users with high-level metrics that are crucial from a business perspective together with extremely detailed insights on containerized services. Docker users see information that is specific to images and containers such as the numbers of images being used, running containers, and per-microservice metrics. Sysdig is started as an open source project that focuses on monitoring microservices, Sysdig sees inside your containers without instrumenting them, says Daniel Liong, a member of the Sysdig product team [12]. What this means is that instead of installing an agent on your Docker host, the Sysdig agent sits at the operating system level so instead of looking from the inside, Sysdig looks at the containers from the outside.

IV. BACKGROUND

A. Virtualization Vs Container based

Current Virtualization techniques can be classified in two major types: container-based virtualization and hypervisor-based virtualization. Container based virtualization at the operating system level, while the hypervisor based system provides virtualization at the hardware level. Each approach has its own significance.

Container-based virtualization application is an OS-level virtualization method for deploying and running distributed applications without launching an entire VM for each application. Instead containers run on a single control host and access a single kernel. Here containers share the same OS kernel as the host, and thus containers can be more efficient than VMs, which require separate OS instances. Containers hold various components such as files, environment variables and libraries. The host OS also constrains the container's access to physical resources such as CPU and memory so a single container cannot consume all of a host's physical resources. Each approach has its own significance. Container based virtualization is lightweight virtualization in which host kernel OS runs various multiple environments. Containers are nothing but virtual environments. Linux-VServer [1], OpenVZ [2], and Linux Container (LXC) [3] are three main approaches for this representation.

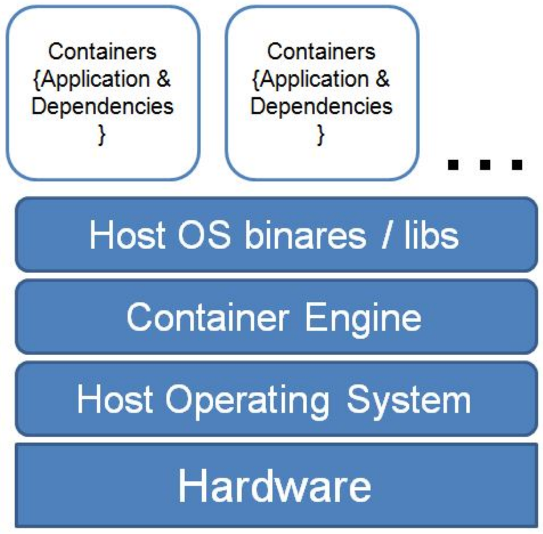

Containers {Application & Dependencies }
Containers {Application & Dependencies }
. . .
Host OS binares / libs
Container Engine
Host Operating System
Hardware

Fig. 1. Architecture of Container-based Virtualization

In contrast ,Hypervisor establishes complete virtual machines (VMs) above the host operating system (Figure 2). Hypervisors use a thin layer of code in software or firmware to allocate resources in real-time. There are two types of hypervisors: Type 1 and Type 2. Type 1 hypervisors run directly on the system hardware. They are often referred to as a "native" or "embedded" hypervisors in vendor literature. Type 2 hypervisors run on a host operating system.

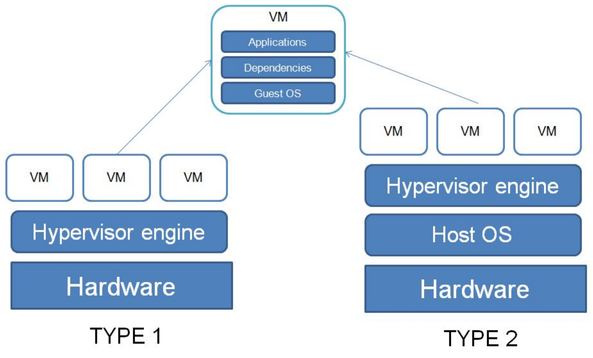


Fig. 2. Architecture of Hypervisor-based Virtualization

When the virtualization movement first began to take off, Type 2 hypervisors were most popular. Administrators could buy the software and install it on a server they already had. Xen [4] is an example of the Type 1 Hypervisor and KVM [5] is of the Type 2 Hypervisor. Since the Type 1 hypervisor does not include an extra layer of the host OS, it provides better performance than the Type 2 hypervisor. Container-based virtualization also offers better performance.

This has been demonstrated by experiments in some recent studies [5,6]. These studies show that the performance of container-based virtualization is better than with hypervisor-based virtualization in most cases, and it is almost as good as native running applications.

B. Docker Overview

Docker is an open source container technology with the ability ”to build, ship, and run distributed applications” [6]. Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment [6]. Although container technologies have been around for more than a decade, Docker - is currently one of the most successful technologies since it comes with new abilities that earlier technologies did not have.

According to industry analyst firm 451 Research, ”Docker is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server. This helps enable flexibility and portability on where the application can run, whether on premises, public cloud, private cloud, bare metal, etc.” [7] .

Docker consist of 2 major components : Docker Engine and Docker hub this will be described in next sections.

C. Docker Engine

The Docker Engine is a lightweight and powerful open source containerization technology combined with a work flow for building and containerizing the applications. Docker Engine is a client-server application with these major components:

- A server which is a type of long-running program called a daemon process.
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client. [8]

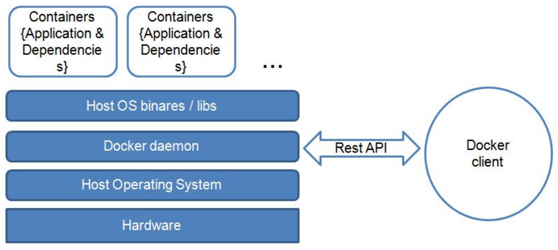


Fig. 3. Architecture of Docker engine

The Docker containers run on top of the Docker daemon which is in charge of executing and managing all of the Docker containers. The Docker client contains a UX for interacting with containers to Docker users, accepts commands from the users and then sends it to the Docker daemon through RESTful APIs

D. Docker Container

A Docker container is a runnable instance of a Docker image. You can run, start, stop, move, or delete a container using Docker API or CLI commands. When you run a container, you can provide configuration metadata such as networking information or environment variables. Each container is an isolated and secure application platform, but can be given access to resources running in a different host or container, as well as persistent storage or databases [8].

A container uses the host machines Linux kernel, and consists of any extra files you add when the image is created, along with metadata associated with the container at creation or when the container is started. Each container is built from an image[8]. The image defines the containers contents, which process to run when the container is launched, and a variety of other configuration details. The Docker image is read-only. When Docker runs a container from an image, it adds a read-write layer on top of the image (using a UnionFS) in which the application runs[8].

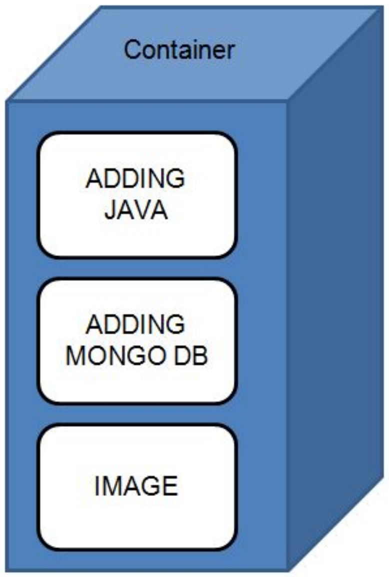


Fig. 4. Docker Container

Docker launches its containers from Docker images. A Docker image is a series of data layers on top of a base im-age. Every Docker image starts from a base im-age, such as CentOS base image or ArchLinux base image. When users make changes to a container, instead of directly writing the changes to the image of the container, Docker adds an additional layer containing the changes to the image.

Docker takes advantages of two Linux features, namespaces and cgroups, to safely create virtual environment for its containers. The cgroups, or control groups, provide mechanism for accounting and limiting the resources which the processes in each container can access. The namespaces wrap the operating system resources into different instances [11].

E. Docker Hub

Docker hub [9] is a central repository of images (both public and private), via which users can share their customized images. Users can also search for published images and download them with the Docker client. Furthermore, users can verify the authenticity and integrity of the downloaded images since Docker signed and verified the images when their owner submitted them to the hub [9].

Docker Hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts [10]. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline [10].

Docker Hub provides a place for the team to build and ship Docker images. Docker Hub configures repositories in two ways:

- Repositories, which allow you to push images from a local Docker daemon to Docker Hub, and.
- Automated Builds, which link to a source code repository and trigger an image rebuild process on Docker Hub when changes are detected in the source code.

V. DATCENTER AND DOCKER USAGE

Nowadays lot of large tech giants like google, facebook and others are deploy millions of docker instances every month. Docker Datacenter (DDC) is a container management and deployment services project from Docker developed to help enterprises get up to speed with their own Docker-ready platforms. Docker describes its Docker Datacenter tool as ”an integrated, end-to-end platform for agile application development and management at any scale.” With Docker Datacenter, organizations are empowered to deploy a Containers as a Services (CaaS) on-premises or in your virtual private cloud. A CaaS provides an IT managed and secured application environment of content and infrastructure where developers can build and deploy applications in a self service manner [13].

VI. OUR APPROACH

There are some inbuild commands and console tools available for monitoring docker instances. Few companies have developed internal tools for monitoring docker instances. Google have developed docker monitoring tool called as cadvisor. This tool monitors the single instance of docker. But this tool does not provide a way to monitor data centers where numerous docker instances are running in parallel. Here, we have developed a push based metrics monitoring system where thousands of docker instances can be monitored from single web based tool. We are storing every metrics from all docker instances to centralized database server. Additionally, we have built a user interface to view these metrics using various graph representations.

VII. SYSTEM ARCHITECTURE

We have designed this system by modular approach. This system involves two independent parts. One part consists of development of framework which will periodically updates the central server with monitoring data from multiple docker instances. Second part is from user point of view which involves the development of web services and user interface for displaying monitoring data.

Overall, this monitoring system mainly involves three main architectural components. They are as follows:

- Monitoring Client
- Web service and Database server
- User Interface

Before we describe all these components in detail, here is the architecture of our system:

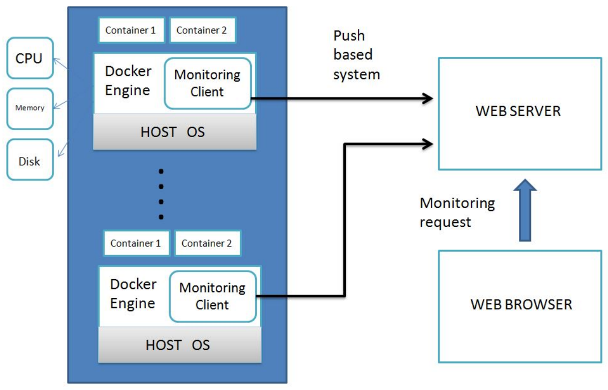


Fig. 5. Complete System Architecture

VIII. SYSTEM DESIGN CHOICES AND COMPARISON

A. Push vs Pull Based System

In software engineering, the pull model and the push model designate two well-known approaches for exchanging data between two distant entities. Normally software systems are stateless as they use the stateless protocol HTTP. So, once a request is served to the client then there is no direct way of notifying him/her about the changes made to the Model. To handle this scenario there are two mechanisms.

B. Push Based

In the push-based system, the client opens a connection to the server and keeps it constantly active. The server will send all new events to the client using that single always-on connection. In other words, the server PUSHes the new events to the client. In Pushed based model, the server pushes data to the client without client explicitly asking for that.

A cron job to push the patch to run servers at regular intervals so that the required estate is accessed continually at intervals. So the advantage of using this technique is whenever the client logs in, he will be running with a new stack and heap. So there is no need for persistent objects on each request or managing a maximum amount of concurrent connections. Additionally, a push-based system puts all the control hands of the central admins. The problem with push-based systems is that you have to have a complete model of the entire architecture on the central push node. You can't push to a machine that you don't know about.

This phenomenon is used to describe the preplanned news, weather or other selected information that is updated on a periodic basis on users desktop interface. Push technology is also a prime feature of Web browsing applications. Synchronous conferencing and instant messaging are typical examples of push services.

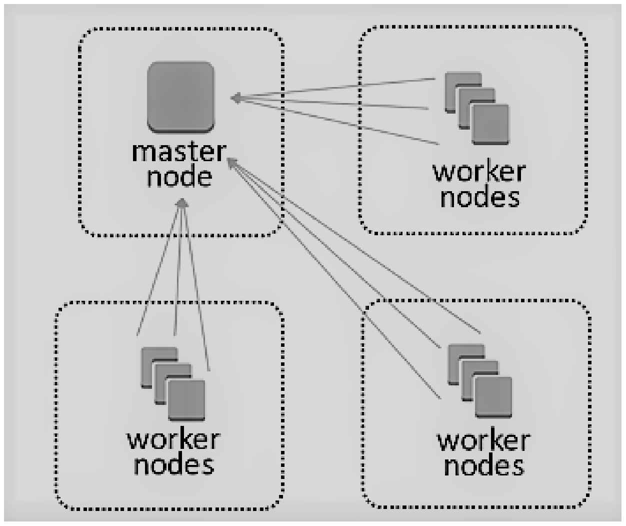


Fig. 6. Push Based Architecture

C. Pull Based

The pull model is based on the request/response paradigm (called data polling, or simply polling, in this management, the client sends a request to the server, then the server answers, either synchronously or asynchronously. This is functionally equivalent to the client pulling the data off the server. In this approach, the data transfer is always initiated by the client, i.e. the manager

In Pull based system, the initial request for data originates from the client, and then is responded to by the server.In this style of network, the client periodically connects to the server, checks for and gets recent events and then closes the connection and disconnects from the server. The client repeats this whole procedure to get updated about new events. In this mode, the clients periodically PULLs the new events from the server. The server or publisher does not send information to the client unrequested.

In a 'pull' system, clients contact the server independently of each other, so the system as a whole is more scalable than a 'push' system. Downloading Web pages via a Web browser is an example of pull technology. Getting mail is also pull technology if the user initiates a request to retrieve it.

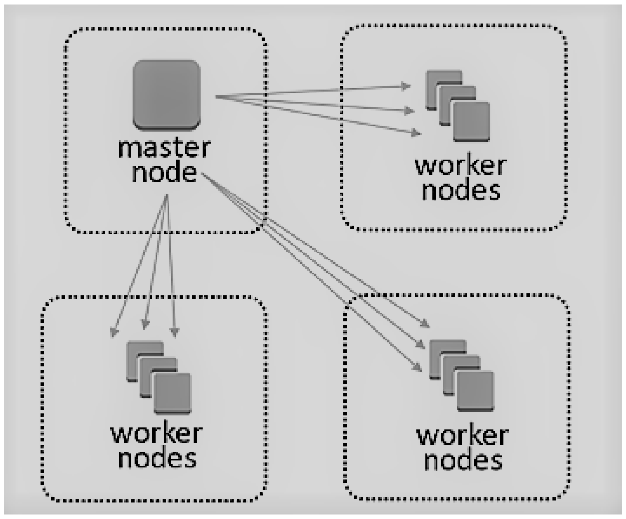


Fig. 7. Pull Based Architecture

Today, most software systems only use one of the methods above and do not implement both push and pull. While in the first look, it may seem good enough as in both ways you can achieve the desired functionality. However, some tasks are better suited by one of them than the other.

The difference is that in push protocols, you get new events literally instantly. But you may experience a small time delay in pull protocols. Although this delay can be overcome by using regular checks for new events so that the time delay is mostly not noticeable. Pull based is more complicated, as the data flow of push-based model is only a subset of that of the pull-based model, hence in result pull-based model contains additional "data request". Additionally,the request for a given session is initiated by the publisher, which is contrasted with pull, where the request for the transmission of information is initiated by the receiver or client.

We cant ignore the fact that push is much better at performing actions and tasks than the pull is, and pull is superior in handling request to show some kind of data.

IX. Why Push Based System for Docker Monitoring

Push technology offers solutions for the information retrieval and browsing off-line problems. That is, once you get the information from the server, you can access this information without being connected. Of course, you should be on-line if you want to get the most recent info. The main advantages of Push technology is convenience along with instant transmission of information and efficient in terms of initiating server connections. Rather than having go hunt for the latest information, it simply arrives automatically on your computer. This saves the users' time since they don't waste a lot of time to search for a specific information. Once it is downloaded, you can browse the content when you are off-line. This is particularly valuable for users with dial-up connections and slow connections. In pull-based mechanism, extra connections are opened, requiring more server resources. This is especially heavy on servers that rely on a one-process-per-connection delivery method. Connection setup causes delays. A new handshake is needed and sometimes a new DNS lookup. While in the push-based system, the server allows administrators to tell the client "you're also gonna need this" which saves multiple network connections and network delays.

A. *Push Based System Approach For Docker*

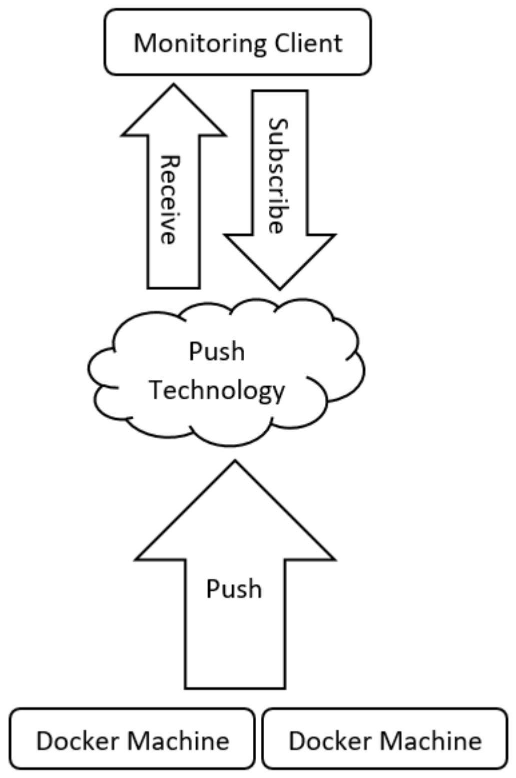


Fig. 8. Docker Push Based Architecture

Above figure represents how the push-based system is implemented for docker monitoring. As explained in the service architecture, each individual docker image will be installed on every docker machine. This image is responsible for fetching the current status of every container installed on docker machine. The status will be in form of current processes running on docker machine, CPU utilization, memory details of a particular docker machine. So, there can be any number of docker machines where this image can be downloaded and executed. Also, this image will be responsible for periodically notifying the current status of docker machine to the centralized server as mentioned in above figure. This centralized server will have a database which will be updated as soon as client docker machine notifies its status to the centralized server. This database will play a role of fetching and showing the current status of a particular docker machine on Web Application. The status fetched from an individual machine will be of particular time duration such as past half and hour to several hours. Additionally, as observed the advantages of the push-based system, this architecture helps in getting the current status of all the docker machine whenever system admins logs into the server i.e. administrator does not require to establish a connection and ask for current status on every docker machine.

Also, if any docker machine is failed to establish the connection with the centralized server, it will not at all affect the overall system and user will be still able to get the monitoring metrics for other docker machines.

B. *Error Handling for Push-Based Architecture*

What if the centralized server which is collecting the status from every docker machine gets down or the connection is failed between a docker machine and a centralized server, to handle these failure cases or erroneous condition lets find out the recovery mechanism to keep the system alive.

The server can reconstruct its state by polling its clients. Since the server may lose the information about the clients, and then it may not be able to poll all the clients. One possible solution is to broadcast a recovery message to notify clients to report their cache information. However, this approach has two problems. First, since we are using pushed based approach, clients need to access the reverse control channel to get the permission to use the uplink channel, it may result in lots of collisions if they respond to the recovery message at the same time. Second, due to disconnections, there is no guarantee that all clients will respond the recovery and then the server may not be able to collect all the necessary information about the clients. Due to the problems or overhead associated with the above solutions, we propose to apply a stepwise approach to address the server failure problem.

We propose a caching based approach for the centralized server to store the incoming monitoring matrics from the individual docker machines. This cache will be useful to serve the request from system administrator as well as to recover the missing requests whenever the centralized server gets fail. This cache system will simply be a database server which will keep on updating along with the database of the centralized server. Every docker machine has to maintain a connection with the cache server as well. The another advantage of having such caching server is that if a load of polling increases on a centralized server, cache serve might help to handle the extra overhead of the load and can serve the requests from the administrator for some time or can store the monitoring statuses from docker machines. This approach is healthy enough to handle the failure conditions as well as load balancing.

C. Communication Framework of Docker machine and Centralized Server

For communication between every docker machine and the centralized server, we are following HTTP. So first let's get introduce to how network communication takes place in the same and its details. It is a short for HyperText Transfer Protocol. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. It's an application layer protocol that is used to transmit virtually all files and other data on the World Wide Web, whether they're HTML files, image files, query results, or anything else. Usually, it takes place through TCP/IP sockets. TCP transport service uses sockets to transfer the data. The client initiates the TCP connection by using sockets on port 80 to the server. Then the server accepts the connection from the client. The client requests with the HTML pages and the objects which are then exchanged between the client browser and web server. After completing the request, the TCP connection is closed. It is a stateless protocol. It does not keep user information about the previous client requests. So, this protocol is simple but if you have to maintain the past client records then it is complex. Since the server will maintain all the client requests and when the server crashes, it is very difficult to get the information back and makes the system very complex.

HTTP clients (such as Web browsers) and Web servers communicate via HTTP request and response messages. The three main HTTP message types are GET, POST, and HEAD as well as few others like PUT, DELETE, CONNECT, etc.

The GET method is used to retrieve information from the given server using a given URI. Requests using this method should only retrieve data and should have no other effect on the data. GET messages sent to a Web server contain only a URL. Zero or more optional data parameters may be appended to the end of the URL. The server processes the optional data portion of the URL if present and returns the result (a Web page or element of a Web page) to the browser. The GET method is the most commonly used. It states that give me this resource. The part of the URL is also called as Request URL. The HTTP is to be in uppercase and the next part denotes the version of HTTP. There is another type HEAD, which is same as GET but returns only HTTP headers and no document body.

A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms. PUT messages place any optional data parameters in the body of the request message rather than adding them to the end of the URL. The Web pages which ask for input from user uses the POST method. The information filled by the web user is uploaded in servers entity body. The typical form submission in POST method. The content type is usually the application/x-www-form-URL encoded and the content-type is the length of the URL-encoded form data.

Considering the current proposed implementation approach for docker monitoring, we are mainly going to use the POST message type for the communication between the docker machine and the centralized server. This API call will be responsible for publishing the current docker machine status periodically to the centralized server. Whenever events for posting the docker status is executed, this post call consumes information such as a name of the container whose status is requested, the time duration for which the monitoring logs are required and in return, it will give the response bundle containing information such as status of the whole system along with the detailed container information. On execution of every POST on docker machine, the response of this call will be processed at the centralized server, which will be made available to serve the later requests from web user interface client. All the container information gave as a response by implemented client is in the form of JSON object.

D. Web Interface For Docker Monitoring

Additionally, we are using the GET call for fetching the monitoring status from the centralized server and rendering it on the user interface. So whenever the user logs in and provides the valid credentials of the server of which he wants to track the status, the GET call will be executed. This call will consume the basic information of the docker system so as to identify it on the centralized system and in response it will return the detail information populated by a docker client. This response will be later rendered on the user interface to represent the information in a proper user readable format. The provided web interface will allow the user to view the information of any docker machine connected to the centralized server. Ultimately this user interface allows the user to remotely fetch the information of any docker machine rather than going into the specific network where the docker actually resides. The monitoring status such as current CPU usage, memory allocation, etc will be shown to the user in form of graphs, charts and several user friendly UI widgets.

X. Implementation Details

As shown in architectural diagram, we have used modular design and development approach for this project. In this section, we will be discussing every development model in detail. This section will cover approaches, challenges and technologies used to develop DocSnap monitoring platform.

A. Monotoring Client

We have developed a monitoring client which is a scheduled cron job implemented as a python script. This job runs at every minute and fetches performance metrics such as Disk Utilization, CPU utilization and memory consume for each container running on docker instance.

Monitoring client mainly built using three main components:

- Google Cadvisor Rest API
- Python Script (Middleware between cadvisor and central Database.)
- Cron job to run Python Script Periodically.

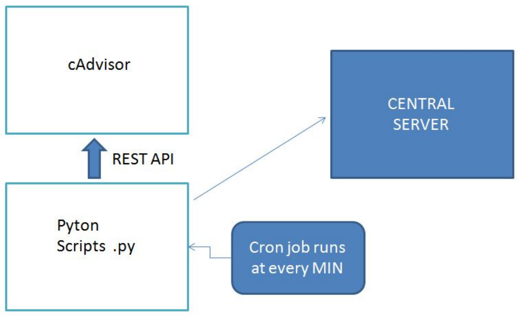


Fig. 9. Monitoring Client

As mentioned above, we have developed a python script which makes REST calls to cadvisor API and receives a JSON response from that. Then this script filters the monitoring metrics and machine information from the response. Filtering process involves extraction of relevant metrics such as CPU and memory metrics. This process involved few challenges as we have to understand the response data returned by cadvisor. Following details will explain the challenges and process we followed:

- CPU Metrics Extraction:
  Cadvisor is a Docker container developed by google to monitor performance of other running containers inside a single Docker instance. Cadvisor monitors all other containers continuously in a tight loop and generates CPU performance statistics at every second per container. Cadvisor generates and represents CPU utilization by series of 10-digit integer numbers where every number represents the CPU consumption time in nanoseconds. This series typically consist of 60 numbers generated per second for duration of 1 minute. Our main challenge was to understand these series of numbers. We figured out this problem by reading GitHub discussions [12] and online documents. We figured out that this long integer is in billionths of core installed on that machine. In simple words, to get the utilization of all CPU cores, one must divide the summation of all integers in series by 1,000,000,000 (1 billion) to get CPU consumption rate. According to the Cadvisor documentation, this series consist of derivative of CPU metrics. So, series of CPU metrics will be rate of change from last metrics to current metrics. After understanding this, we decided that we will be storing a mean derivative at every minute. We designed a custom approach by which we calculate the difference between maximum and minimum derivative and then divide this number by number of derivatives in series which typically consist 60 entries. After calculating mean derivative, we divide this number by 1,000,000,000 which give us the utilization of all CPU cores. To calculate CPU utilization per core, we divide this number by number of cores installed on that Docker instance. In this way, we derive and calculate CPU utilization per minute per container.
- Memory Metrics:
  Similar to CPU monitoring, Cadvisor generates memory utilization statistics at every second. We are calculating mean for series of memory consumptions happened in a minute time. Cadvisor API returns series of memory metrics in terms of bytes that RAM have consumed in last 1 minute. We are converting these bytes into Megabytes to represent on the front end. Advisor returns all other statistics such as Disk utilization, network metrics and some other system performance metrics. But in current prototype, we are only concentrating on CPU and memory metrics.
  Next important point about monitoring client is the development of cron job. We have designed and developed this monitoring client using Python 2.7 and crontab which is inbuilt Linux tool to execute periodic jobs. We have configured this cron job to run at every minute and send these statistics to our central database server. We have implemented HTTP post call to send these statistics over the network to DB server. Here every Docker instance will push these metrics to central server. This is the reason our architecture is defined as push based system. Details of HTTP post call and push based will be discussed in next point.

B. Web services and Database server

We have developed a client server web application using three tier architecture which mainly involve three tiers of design.

- Presentation tier
- Logic tier
- Data tier

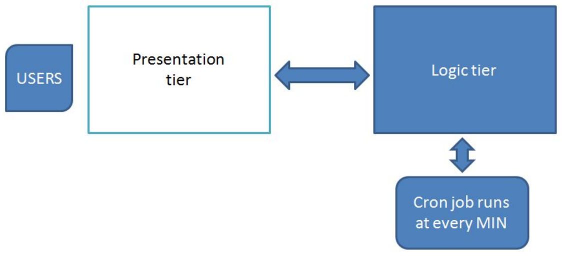


Fig. 10. Web Service and Data Center

In this section, we will be discussing logic tier and data tier in detail.

- Logic Tier
  Logic tier sits between presentation and data layer. This layer mainly composed of all application logic and processing required for data before sending it to the front end. We have designed our logic tier using Pylons web framework. Pylons is an open source software package developed in Python. Our web application mainly consists of two endpoints. Both these endpoints are designed using REST protocol. One endpoint has been designed for receiving data posted from push based system. This endpoint extracts the data from HTTP body and loads that data into JSON format. This JSON data is separated in various hash maps and stored in temporary variables. After that it has been stored in database. Second endpoint is designed to provide metrics for input Docker instance. This endpoint returns the output in JSON and returns it to frontend. This endpoint also returns machine details such as system UUID, number of cores, total RAM, mac address etc. This endpoint also returns the container details which include name and image details.

- Data tier
  Data tier mainly designed to store all the data generated in monitoring system. We have used MongoDB NoSQL database to store all metrics and machine data. We have designed collection called as MonitoringStats which has fields such as containers, machine, machineid etc. We are using pymongo which is database driver which communicates with MongoDB from python environment. MongoDB communicates with external environment over TCP port. We choose MongoDB over other databases because MongoDB scales and NoSQL which gives us the ability to store dynamic data. As we are dealing with machine metrics and log data, most of the time schema for that data will be dynamic. Thats why we choose MongoDB to store data.

C. User Interface

We have designed user interface which provides a single platform for sysadmins to monitor all Docker instances. This user interface is nothing but the presentation layer of our system. User interface mainly displays some key information of the machine also represents memory and CPU utilization using graphs.

We have used HTML, JavaScript and CSS frameworks for development of frontend. We have also used few open source JavaScript libraries for the implementation of frontend. We have used bootstrap.js and Highcharts.js to provide crisp User Interface. We dynamically populate graphs from data returned by server using Highcarts.js. Detail screenshots and results will be represented in Results section.

XI. EXPERIMENT SETUP

We have tested this setup on local machine by configuring local machine as docker instance. We have installed two containers on this docker instance.

A. Ubuntu Container

This container runs light weight Ubuntu OS. We have executed following command to create maximum load on docker instance and generate monitoring stats.

B. Cadvisor Container

This container run cadvisor API service in background. It generates fair amount of CPU and memory load.

After this setup, we have monitored stats for this local instance from User Interface we have developed. Detail screenshots and explanation is in next section.

XII. ISSUES ENCOUNTERED

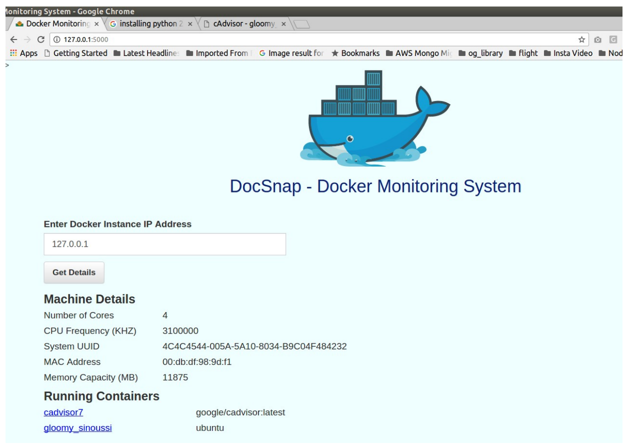
A. Choice for Metrics Storage

As we have specified, we have chosen MongoDB as a primary data storage for storing performance metrics. But in early stages we opted for Redis which is a inmemory data storage system. It stores the data in key value pairs and designed for quick access time. Redis stores all hot data that is frequent metrics in main memory RAM. But we started putting storing data in redis, we noticed the spike in CPU and memory utilization of our backend server. After debugging the issue with linux commands such as top, free and ps, we figured out that Redis was causing spike in memory due to heavy data read and write. So, we decided to scrap the Redis and used persistent storage as MongoDB.

B. Understanding CPU Metrics

This point is already covered in section X under point A with title CPU Metrics Extraction.

XIII. RESULT AND ANALYSIS



Enter Docker Instance IP Address

127.0.0.1

Machine Details

Number of Cores	4
CPU Frequency (KHZ)	3100000
System UUID	4C4C4544-005A-5A10-8034-B9C04F484232
MAC Address	00:db:df:98:9d:f1
Memory Capacity (MB)	11875

Running Containers

cadvisor7	google/cadvisor:latest
gloomy_sinoussi	ubuntu

Above screenshot is taken from the actual project demo, it's a main page or the application where user can provide

Fig. 11. DockSnap Main Screen

the IP address of the docker instance as a credential of which he/she wants to check the current status. On providing valid ip address, the server returns the complete information along with monitoring status per of all the docker instances.

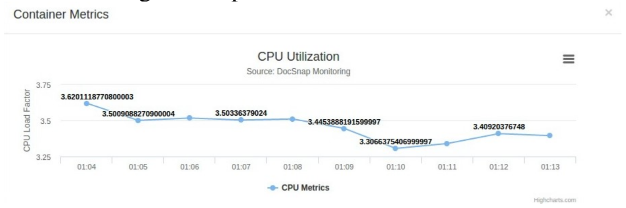


Fig. 12. CPU Utilization Metrics Result

Above screenshot represents the current CPU monitoring status of the selected container. User can view these metrics by selecting any of the container enlisted on the main screen. These CPU metrics are generated or fetched for the last 10 mins and shown on a interval of per minute.

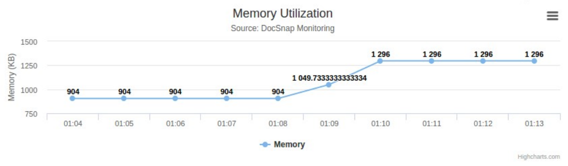


Fig. 13. Memory Utilization Result Metrics

Above screenshot represents the current Memory utilization status of the selected container. User can view these status by selecting any of the container enlisted on the main screen. These memory metrics are generated or fetched for the last 10 mins and shown on a interval of per minute.

XIV. TEAM ROLES

Every team member was involved in design and development of docker monitoring application. Throughout out project implementation, sub-tasks were equally divided based on individual's expertise. Along with separate tasks, there were some of the modules on which everyone needed to work collaboratively. These modules include actions from exploring docker and its container-based application, understating the execution and usage of docker commands, complete integration testing of the application and lastly in the documentation of the project. Following are the brief details of individual team-mates contribution in the project:

A. Pratik Bhangale

Responsible for exploring the API provided for docker monitoring by Cadvisor and implementing the implementing the essential calls as per requirement. Along with API implementation, Pratik was also responsible for designing and development of user interface of this monitoring tool.

B. Vishal Rathod

Responsible developing the cron job which will be executed periodically to push the necessary monitoring status from every docker machine. These scripts and complete application are exposed as a single image so that it can be easily installed on docker machines. Along with this, Vishal was also responsible for designing and development of the database of the centralized server.

C. Mayur Pate

Responsible for development of the python application which integrates the API implementation and the database connectivity. This application is responsible for updating the centralized server and handling the error situations. Along with this, Mayur was also responsible for the integration of pylon application with the user interface.

XV. FUTURE SCOPE

A. Scaling

Currently this system is storing all monitoring stats in MongoDB via single application server. This system need to be tested in production environment to see how it scales. Scaling will certainly involve issues related with web server, database server and caching system.

B. Data Storage - Hadoop

We will be dealing with large amount of data generated from thousands of docker instances. Down the line, we will need large data storage such as Hadoop to store metrics data. Some advance frameworks such as Spark, Pig etc will required for data retrieval.

XVI. CONCLUSION

Docker is the latest hot trend in the technology domain. All the tech giants are started using docker for building their systems. Along with such demands, managing docker system are even more essential. Hence, considering current technological need, we have proposed an important solution to monitor the docker current performance using a simple web based approach. DockSNAP is currently targetting only those components which requires careful management operation from system administrators. DockSNAP helps in reducing manual efforts of accessing every docker machines via physically logining into every system, it provides a web based centralized console to access and monitor every docker machine from single remote system. The biggest advantage of this remote accessibility allows system administrator to monitor several thousands of docker systems in a datacenter from a single computer. These detail monitoring information provides Containers, CPU, Memory, etc information of the whole docker system.

References

[1] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: A scalable, high- performance alterna- tive to hypervisors. In Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pages 275287, USA, 2007. ACM.

[2] OpenVZ. http://openvz.org/. [Accessed 30 September 2014].

[3] LXC. https://linuxcontainers.org/. [Ac- cessed 30 September 2014].

[4] B. R. Anderson, A. K. Joines, and T. E. Daniels. Xen worlds: Leveraging virtualization in distance educa- tion. In Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 09, pages 293297, New York, NY, USA, 2009. ACM.

[5] N. Regola and J.-C. Ducom. Recommendations for vir- tualization technologies in high performance comput- ing. In 2010 IEEE Second International Conference on Cloud Computing Technology and Science (Cloud- Com), pages 409416, Nov. 2010.

[6] What is docker? https://docker.com/whatisdocker/. [Accessed 15 November 2014].

[7] Noyes, Katherine (1 August 2013). &quot;Docker: A &39;Shipping Container&39; for Linux Code&quot;. Linux.com. Retrieved 2013-08-09.

[8] https://docs.docker.com/engine/understanding-docker/

[9] Docker hub. https://hub.docker.com/. [Accessed 30 September 2014].

[10] Overview of Docker hub - https://docs.docker.com/docker-hub/

[11] https://www.researchgate.net/publication/270906436 Analysis of Docker Security

[12] Sisdog is open source, system exploration - http://www.sysdig.org/

[13] Docker data centre https://blog.docker.com/2016/02/docker-datacenter-caas/

[14] http://www.cs.bilkent.edu.tr/ david/cs533/barisuz/webpages/push04.html

[15] https://www.lifewire.com/hypertext-transfer-protocol-817944

[16] HTML - CSS tutorials http://www.w3schools.com/css/

[17] Python tutorials - https://docs.python.org/3/tutorial/

[18] How to write Cron files - https://corenominal.org/2016/05/12/howto-setup-a-crontab-file/

[19] Java script Tutorials - http://www.w3schools.com/js/

[20] https://docs.docker.com/engine/installation/linux/ubuntulinux/