Grenze International Journal of Engineering and Technology, Special issue

Containerized Web Proxy Server for Campus Network

Bharanidharan P[1], Sathish Chandra G[2] and Karthick K[3]
[1]Kongunadu College of Engineering and Technology/ CSE, Trichy, Tamilnadu, India
Email: bharani3b@gmail.com
[2] Kongunadu College of Engineering and Technology/ CSE, Trichy, Tamilnadu, India
Email: sathishchandra98@gmail.com
[3] Kongunadu College of Engineering and Technology/ AP -CSE, Trichy, Tamilnadu, India
Email: karthivel.me@gmail.com

***Abstract*—Project aims on deploying a containerized proxy server with features such as proxy, cache, web filtering and access control. The existing systems provide installation of proxy servers on server systems which requires manual proxy configuration on client browsers. This server provides proxy operations without any manual configuration on the client systems. This server reduces bandwidth usage and improves response times by caching and reusing frequently-requested web pages. This server will have extensive access controls and makes a great server accelerator. This server offers a surprisingly easy way to save on bandwidth, as well as provides an easy way to force authentication to be required in order to obtain outbound access to traffic. This server can affect multiple connections at the same time without their speeds taking a serious hit. All of this comes in a docker based container.**

***Index Terms*— Server Accelerator, Proxy, Caching, Access Control.**

I. INTRODUCTION

A. Caching

Caching is the term for storing reusable responses in order to make subsequent requests faster. There are many different types of caching available, each of which has its own characteristics. Application caches and memory caches are both popular for their ability to speed up certain responses.

Web caching, the focus of this guide, is a different type of cache. Web caching is a core design feature of the HTTP protocol meant to minimize network traffic while improving the perceived responsiveness of the system as a whole. Caches are found at every level of a content's journey from the original server to the browser.

Web caching works by caching the HTTP responses for requests according to certain rules. Subsequent requests for cached content can then be fulfilled from a cache closer to the user instead of sending the request all the way back to the web server.

B. Cache server

A cache server is a dedicated network server or service acting as a server that saves Web pages or other Internet content locally. By placing previously requested information in temporary storage, or cache, a cache server both speeds up access to data and reduces demand on an enterprise's bandwidth. Cache servers also allow users to access content offline, including rich media files or other documents. A cache server is sometimes called a "cache engine."

Grenze ID: 01.GIJET.5.1.30

A cache server is almost always also a proxy server, which is a server that "represents" users by intercepting their Internet requests and managing them for users. Typically, this is because enterprise resources are being protected by a firewall server. That server allows outgoing requests to go out but screens all incoming traffic. A proxy server helps match incoming messages with outgoing requests. In doing so, it is in a position to also cache the files that are received for later recall by any user. To the user, the proxy and cache servers are invisible; all Internet requests and returned responses appear to be coming from the addressed place on the Internet. (The proxy is not quite invisible; its IP address has to be specified as a configuration option to the browser or other protocol program.)

C. Caching Architectures

Caching Architectures Caches sharing mutual trust may assist each other to increase the hit rate. A caching architecture should provide the paradigm for proxies to cooperate efficiently with each other. Two common approaches to implement a large-scale cache cooperation scheme are hierarchical and distributed caching.

D. Hierarchical Caching

With hierarchical caching caches are placed at different network levels. At the bottom level of the hierarchy there are client caches. When a client cache does not satisfy a request, the request is redirected to the institutional cache. If the document is not present at the institutional level, the request travels to the regional cache, which in turn forwards unsatisfied requests to the national cache. If the document is not present at any cache level, the national cache contacts directly the origin server. When the document is found, either at a cache or at the origin server, it travels down the hierarchy, leaving a copy at each of the intermediate caches. Further requests for the same document travel up the caching hierarchy until the request finds the document. There are several problems associated with a caching hierarchy: i) every hierarchy introduces additional delays, ii) higher level caches may become bottlenecks and have long queuing delays, and iii) several copies of the same document are stored at different cache levels.

E. Distributed Caching

With distributed caching no intermediate caches are set up and there are only caches at the edge of the network that cooperate to serve each other's misses. In order to decide from which cache to retrieve a miss document, all caches keep meta-data information about the content of every Session . Most of the traffic flows through low network levels, which are less congested and no additional disk space is required at intermediate network levels. In addition, distributed caching allows better load sharing and is more fault tolerant. However, a large-scale deployment of distributed caching may encounter several problems such as high connection times, higher bandwidth usage or administrative issues.

F. Developing a caching strategy

In a perfect world, everything could be cached aggressively and your servers would only be contacted to validate content occasionally. This doesn't often happen in practice though, so you should try to set some sane caching policies that aim to balance between implementing long-term caching and responding to the demands of a changing site.

Common Issues

There are many situations where caching cannot or should not be implemented due to how the content is produced (dynamically generated per user) or the nature of the content (sensitive banking information, for example). Another problem that many administrators face when setting up caching is the situation where older versions of your content are out in the wild, not yet stale, even though new versions have been published.

These are both frequently encountered issues that can have serious impacts on cache performance and the accuracy of content you are serving. However, we can mitigate these issues by developing caching policies that anticipate these problems.

General Recommendations

While your situation will dictate the caching strategy you use, the following recommendations can help guide you towards some reasonable decisions.

There are certain steps that you can take to increase your cache hit ratio before worrying about the specific headers you use. Some ideas are:

Establish specific directories for images, css, and shared content: Placing content into dedicated directories will allow you to easily refer to them from any page on your site.

Use the same URL to refer to the same items: Since caches key off of both the host and the path to the content requested, ensure that you refer to your content in the same way on all of your pages. The previous recommendation makes this significantly easier.
Use CSS image sprites where possible: CSS image sprites for items like icons and navigation decrease the number of round trips needed to render your site and allow your site to cache that single sprite for a long time.
Host scripts and external resources locally where possible: If you utilize javascript scripts and other external resources, consider hosting those resources on your own servers if the correct headers are not being provided upstream. Note that you will have to be aware of any updates made to the resource upstream so that you can update your local copy.
Fingerprint cache items: For static content like CSS and Javascript files, it may be appropriate to fingerprint each item. This means adding a unique identifier to the filename (often a hash of the file) so that if the resource is modified, the new resource name can be requested, causing the requests to correctly bypass the cache. There are a variety of tools that can assist in creating fingerprints and modifying the references to them within HTML documents.
In terms of selecting the correct headers for different items, the following can serve as a general reference:
Allow all caches to store generic assets: Static content and content that is not user-specific can and should be cached at all points in the delivery chain. This will allow intermediary caches to respond with the content for multiple users.
Allow browsers to cache user-specific assets: For per-user content, it is often acceptable and useful to allow caching within the user's browser. While this content would not be appropriate to cache on any intermediary caching proxies, caching in the browser will allow for instant retrieval for users during subsequent visits.
Make exceptions for essential time-sensitive content: If you have content that is time-sensitive, make an exception to the above rules so that the out-dated content is not served in critical situations. For instance, if your site has a shopping cart, it should reflect the items in the cart immediately. Depending on the nature of the content, the no-cache or no-store options can be set in the Cache-Control header to achieve this.
Always provide validators: Validators allow stale content to be refreshed without having to download the entire resource again. Setting the Etag and the Last-Modified headers allow caches to validate their content and re-serve it if it has not been modified at the origin, further reducing load.
Set long freshness times for supporting content: In order to leverage caching effectively, elements that are requested as supporting content to fulfill a request should often have a long freshness setting. This is generally appropriate for items like images and CSS that are pulled in to render the HTML page requested by the user. Setting extended freshness times, combined with fingerprinting, allows caches to store these resources for long periods of time. If the assets change, the modified fingerprint will invalidate the cached item and will trigger a download of the new content. Until then, the supporting items can be cached far into the future.
Set short freshness times for parent content: In order to make the above scheme work, the containing item must have relatively short freshness times or may not be cached at all. This is typically the HTML page that calls in the other assisting content. The HTML itself will be downloaded frequently, allowing it to respond to changes rapidly. The supporting content can then be cached aggressively.
The key is to strike a balance that favors aggressive caching where possible while leaving opportunities to invalidate entries in the future when changes are made.

II. Existing Systems

- In general every small scale network consists of a WAN network which acts as a outbound connection to the internet.
- For example: If a student in the campus network browse for a website named www.domain.com ,then he/she has to pass through local area network (LAN) and through the outer network (WAN) .
- The above process remains same for every website the student visits.
- This results in heavy usage of data for the campus network and lower utilization of the local area network (LAN).
- Caching servers are used in some areas but yet they don’t provide exact caching mechanisms.

III. Proposed Work

- This project provides a proxy server which can act both as a proxy and a caching server at the same time.

- There is no client side configuration required for this implementation.
- This web proxy server caches the requests that are going through the LAN and provides the content from the local server itself.
- Web filtering feature in this project provides filtering of contents that are identified harmful using the proxy module.
- This project also filters users accessing harmful websites

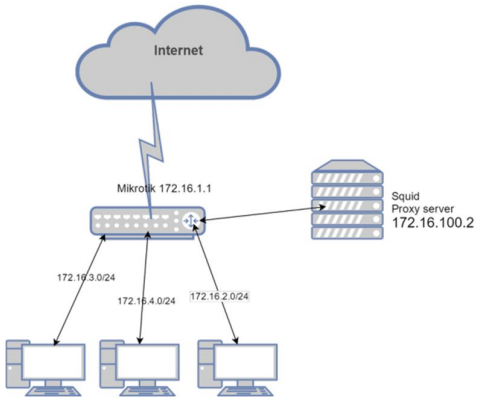

Fig 1. System Architecture

IV. CONFIGURATION

To configure cache server, adjust the directives in the configuration file. Squid is normally configured according to the requirements of a given network using the command line and editing the Squid configuration file, located at /etc/squid/squid.conf, which contains recommended minimum configuration.

A. Basic configuration

1. Backup the original config file.
mv /etc/squid/squid.conf /etc/squid/squid.conf.org
2. Create a new /etc/squid/squid.conf file with the following contents. Edit the Access Control List (ACL) line for mynetwork to define source network for your local network. This is the network where client systems use the Squid server as their proxy.
4. Start the service and enable it on boot:
~]# systemctl enable squid ~]# systemctl start squid
4. If firewall is enabled, allow the Squid port.
~]# firewall-cmd --add-port=3128/tcp --permanent
5. Configure your web browser to use the proxy. This depends on the browser you use and its version. For example, to configure Firefox version 46.0.0:

B. Procedure

Configuring Firefox with Proxy
1. In the Firefox menu located in the top right corner, select Preferences, from the tabs on the left, select Advanced, and then select Network from the tabs located on the top bar.
2. In the Connection section, open Settings.
3. In the new window that opens up, tick Manual proxy configuration and enter the proxy server that you are connecting to in the HTTP Proxy field. If you need to enter a specific port, enter it into the Port field.

C. Configuring Squid as an HTTP proxy server

1. Add the following lines to the top of the /etc/squid/squid.conf file replacing the example IP address :
cache_dir ufs /var/spool/squid 500 16 256 acl my_machine src 192.0.2.21 # Replace with your IP address http_access allow my_machine
2. Create cache directories using the following command:
~]# systemctl restart squid
Squid now starts listening on port 3128 (default) on all network interfaces on the machine.

4. Configure your browser, for example Firefox, to use Squid as an HTTP proxy server with the host as the IP address of the machine and port 3128: for details, see Procedure 15.2, “Configuring Firefox with Proxy”

D. Setting the HTTP Port

The http_port directive is used to specify the port where Squid will listen for client connections. The default behavior is to listen on port 3128 on all the available interfaces on a machine. You can force Squid to listen on multiple interfaces and on different ports, on different interfaces.

Specifying the HTTP Port

Open /etc/squid/squid.conf and edit the respective line. In this example, Squid is set up to listen on port 8080.

Squid normally listens to port 3128 http_port 8080

The Squid server can listen on multiple ports at the same time.

Specifying Two or More Ports

With the following setting, Squid listens on both port 8080 and port 9090:

http_port 8080 http_port 9090

Setting IP addresses

The following command instructs Squid to listen on port 3128 on the interface with the IP address 192.0.2.25:

http_port 192.0.2.25:3128

In addition, you can specify http_port by using host name and port combination. The host name will be translated to an IP address by Squid, which will then listen on port 8080 on that particular IP address.

http_port myproxy.example.com:8080

Another aspect of the http_port directive is that it can take multiple values on separate lines. The following lines will trigger Squid to listen on three different IP addresses and port combinations. This is generally helpful when you have clients in different LANs, which are configured to use different ports for the proxy server. Edit the /etc/squid/squid.conf file as follows:

http_port 192.0.2.25:8080 http_port lan1.example.com:3128 http_port lan2.example.com:8081

E. ACLs and HTTP access control

Access Control Lists (ACLs) are the base elements for access control and are normally used in combination with other directives, such as http_access, to control access to various Squid components and web resources.

F. Squid Authentication

For authentication, the Squid source code connects with a few authentication back ends, also called helpers, such as SMB (SMB server like Windows NT or Samba), DB (an SQL database), or LDAP (Lightweight Directory Access Protocol). Users are authenticated if Squid is configured to use proxy_auth ACLs.

Instruct Squid which authentication helper program to use with the auth_param directive in /etc/squid/squid.conf. Specify the name of the program and any command line options if necessary.

auth_param scheme parameter [setting]

Adding proxy_auth ACLs

Add proxy_auth ACL entries to your Squid configuration by specifying individual user names. In this example, users named lisa, sarah, joe, and frank are allowed to use the proxy at all times. Other users are allowed only during daytime hours.

acl foo proxy_auth REQUIRED acl bar proxy_auth lisa sarah frank joe acl daytime time 08:00-17:00 http_access allow foo daytime http_access allow bar http_access deny all.Mostly, Squid is used for blocking access to certain web content. Typically, either certain ports are blocked or particular web sites.

G. Restricting Access by Blocking a Port

By this method, also called port filtering, you can block a specific port number with the Squid proxy server. Doing so, you can restrict the use of some protocols, services, websites, applications. For example, to block FTP traffic, it is enough to block port 21/TCP. In the same way you can block all HTTPS sites by blocking port 443/TCP.

H. Blocking Port Numbers

1. Log in as the root user and open the Squid configuration file:

~]# vi /etc/squid/squid.conf

2. Block ports using ACLs.

acl Bad_ports port 443 #(create acl for port 443/tcp)

4. Save the changes.

4. Restart Squid to apply the new configuration:
~]# service squid reload

I. Restricting Access by Blocking Specific Sites or Addresses

Configure Squid for your network to disable access to specific sites.

J. Blocking a Specific Website

1. Enable access to Squid on your network. Open the /etc/squid/squid.conf file and search for "Access Controls". Scroll down to INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS. Make sure you adapt the list to your internal IP networks from where browsing should be allowed. In this example, ACL allows access from the local networks 192.168.1.0/24 and 192.168.2.0/24.
INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS acl our_networks src 192.168.1.0/24 192.168.2.0/24 http_access allow our_networks
2. Create a file containing a list of sites you want to block. Name the files, for example, /usr/local/etc/allowed-sites.squid and /usr/local/etc/restrictedsites.squid.
~]# cat /usr/local/etc/allowed-sites.squid www.redhat.com fedoraproject.org
~]# cat /usr/local/etc/restricted-sites.squid www.badsites.com illegal.com
These can then be used to block the restricted sites.
~]# vi /etc/squid/squid.conf
acl our_networks src 192.168.1.0/24 192.168.2.0/24 acl GoodSites dstdomain "/usr/local/etc/allowed-sites.squid" acl BadSites dstdomain "/usr/local/etc/restricted-sites.squid"
http_access allow our_networks http_access deny BadSites http_access allow home_network business_hours GoodSites
Save and close the file.
4. Restart the Squid proxy server:
~]# systemctl restart squid
5. Configure your web browser to use the DNS name or IP address of your Squid server and match the running port.

V. Conclusions

We developed a containerized cache server which provides basic caching of web documents, static contents, media files. This application has a minimal size when compared to other cache server applications and provides high efficiency. This application is highly configurable and can be configured as per user's interest.The future enhancements of this application includes development of UI (user interface) for maintaining the cache server, reducing the size of the application to a far minimal level. The future enhancements will also include DNS based caching on the same cache server which will provide more efficient site accessing and reduces bandwidth even more.

References

[1] Avram, Abel, "Docker: Automated and Consistent Software Deployments". August 9, 2013.
[2] C.-Y. Chang and M.-S. Chen, "On exploring aggregate effect for efficient cache replacement in transcoding proxies," IEEE Trans. on Parallel and Distributed Systems, 14(6), 2003, 611–624.
[3] Christopher M. Judd ,"Getting Started With Docker", Nov 3, 2015.
[4] Dan Walsh,"Yet Another Reason Containers Don't Contain: Kernel Keyrings",April 13, 2015.
[5] S. G. Dykes and K. A. Robbins, "A viability analysis of cooperative proxy caching," in Proc. of IEEE Infocom 2001, April 2001.
[6] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol," In Proceedings of SIGCOMM98, Extended version available as technical Report, 1998, pp. 1361.
[7] Z. Liang, "Transparent Web Caching with Load Balancing," Master Thesis in Queens University, 2001.
[8] "Squid FAQ: About Squid". Retrieved 2007-02-14.
[9] Swan, Chris,"Docker drops LXC as default execution environment", January 20, 2015.
[10] B. Zenel, ―A general purpose proxy filtering mechanism applied to the mobile environment,‖ Wireless Networks, 5(5), 1999, 391–409.
[11] Asokan R, Natarajan AM, Venkatesh C. Ant based dynamic source routing protocol to support multiple quality of service (QoS) metrics in mobile ad hoc networks. International Journal of Computer Science and Security 2008;2(3).
[12] Ranjithkumar, S. and Thillaiarasu, N., 2015. A Survey of Secure Routing Protocols of Mobile AdHoc Network. SSRG International Journal of Computer Science and Engineering (SSRG-IJCSE)–volume, 2.

[13] D. Padmini Bai and P. Preethi , “Security Enhancement of Health Information Exchange Based on Cloud Computing System”, International Journal of Scientific Engineering and Research, pp. 79-82, Volume 4 Issue 10, October 2016.
[14] Bhuvaneshwari .M; Dr.J.Yogapriya. 2016, “Decentralized, Energy-Efficient,Low Latency and Less Homogeneous Settings based Workload Management in Enterprise Clouds”, International Journal of Scientific Engineering and Research, ISSN:2347-3878,Vol.4,Issue No.10,PP.57-62.
[15] Christo Paul.E Preethi.P, Baskaran.G, An Eyes-Free Model Implementation: Voice Based Optical Character Recognition for Mobile Devices, International Journal of Advanced Research in Computer Science & Technology, Volume 2, Issue 1, March 2014.
[16] Saravanan Kumarasamy and Dr.R.Asokan An Efficient Detection Mechanism for Distributed Denial of Service (DDoS) Attack. International Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol.1, No.5, December 2011.
[17] Dr.Raghavendran P.S., Dr.Asokan R., Vishnupriya M., Enhancing the security against Flooding Attack in MANET for Medical Application, Asian Journal of Research in Social Sciences and Humanities, Vol. 6, Issue 6, 2016.
[18] R.Pavithra, Dr.R.Asokan, K.Baskar,Improving Privacy And Dynamic Updation Using Ranked Search Over Outsourced Cloud Data, International Research Journal of Engineering and Technology (IRJET), Volume: 03 Issue: 10, Oct-2016.
[19] Saravanan, K; Asokan, R; Venkatachalam, K., Neuro- Fuzzy Based Clustering of Distributed Denial of Service (DDoS) Attack Detection Mechanism, International Information Institute (Tokyo). Information; Koganei Vol. 16, Iss. 11, (Nov 2013).
[20] Vellingiri, J.; Balambigai, S.; Saravanan, K.; Asokan, R., Secure Real Time Web Based Electrocardiogram Monitoring System for Improved Healthcare, Journal of Medical Imaging and Health Informatics, Volume 6, Number 3, June 2016, pp. 774-778(5).
[21] Yogapriya, J., Saravanabhavan, C., Asokan, R., Vennila, I., Preethi, P., Nithya, B. 2018. A Study of Image Retrieval System Based on Feature Extraction, Selection, Classification and Similarity Measurements. Journal of Medical Imaging and Health Informatics, 8(3), 479-484.