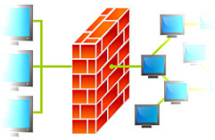
Publishing internal services behind a NAT



Description	Secure, scalable and easy-to-configure HTTP(S) services.
Period	Summer 2014
Languages & Libs	Bash
Tags	Nginx, NAT, SSH

Even in desktop applications it is becoming more and more common to provide a HTTP based APIs or full user interfaces. For example BitTorrent's μTorrent and BitTorrent Sync don't have any built-in UI, and instead users just head with their preferred internet browser to http://localhost:8080 or http://localhost:8888. However they typically lack HTTPS encryption and each port needs to be configured to the NAT router individually. This solution uses a Nginx instance on a virtual machine to provide a HTTPS reverse proxy to all these services in a single port under different sub-domains.

My basic requirement was to make internal HTTP services accessible from external networks by an easy-to-configure and relatively secure solution. I also wanted to minimize configuration on the router side, because it doesn't scale nicely to dozens of ports and a few machines. It is also unable to add HTTPS wrapping on HTTP services. Also the true server shouldn't be directly visible to the outside world, to mitigate the risk of Heartbleed-like bugs and have the configuration hardened for this purpose. Within the internal network it is fine to use password-only SSH authentication, but logins originating from the external network should be allowed only by public/private key authentication.

This solution requires only two ports to be configured to the NAT router: 443 for HTTPS and some arbitrary port for SSH. All traffic is routed to a virtual machine (4 GB disk and 512 MB RAM is sufficient), which only needs sshd and Nginx services running. Currently the Nginx is serving a self-signed certificate, which is secure as long as you check that the SHA-256 checksum matches with the cert you have generated.

The overall network and service structure is shown in Figure 1. The virtual machine is configured to get its own IP from the router's DHCP service. HTTPS is terminated at the virtual machine's Nginx, and based on the request's sub-domain it is proxy-passed to the relevant service. Services have their own internal domains which are configured in the hosts file (a more advanced alternative would be to run an internal DNS service). The actual service can locate in any other machine within the network as well, if there are multiple servers at work. An additional benefit is the possibility of adding a common HTTP Basic authentication on virtual machine's Nginx. So far I have been very happy with this set-up. I can also run optional services such as the Squid HTTP proxy.

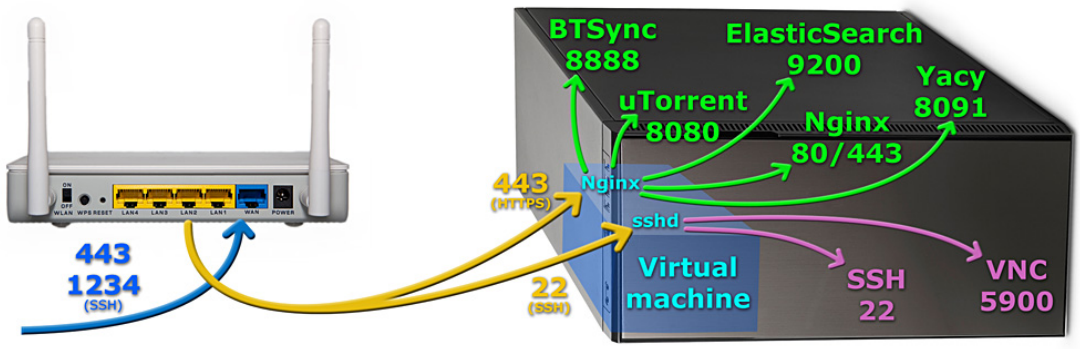


Figure 1: The network topology and some common services' port numbers. Services with HTTP access are shown in green and are served by Nginx reverse proxy, and other services are made visible by SSH tunneling and port-forwarding. The virtual machine is the only machine visible to the outside world, and it accepts SSH logins only by a public/private key.