Vast – Home Assignment

*Shira Benbenisti*

[*Shira1benbenisti@gmail.com*](mailto:Shira1benbenisti@gmail.com)

*054-8181557*

Section 1: System Setup

1. Steps taken for installation and configuration:  
   I wrote two Dockerfiles: one to configure the NFS server and one for the NFS client. Then, I created a docker-compose.yml file that connects both containers (server and client) into the same Docker network, so they can communicate easily.
2. Verification commands and outputs ensuring services restart after reboot:  
   In the docker-compose.yml file, I used **volumes** to map a folder from the host machine into the container. This way, even after a reboot or container restart, the shared data and configurations are preserved, and the services can start again without losing anything.

Section 2: NFS Configuration & Testing

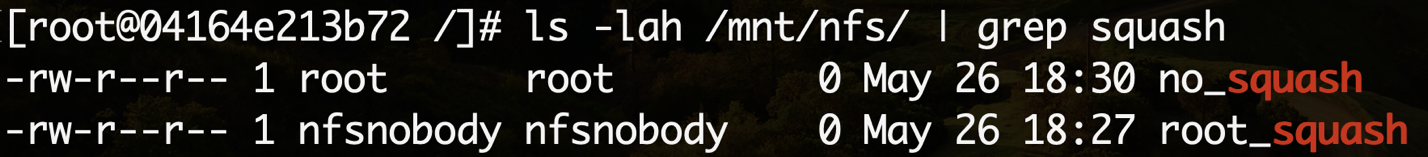
1. Detailed commands used for configuration:

All the commands used are included in the attached README.md file.

1. Explanation of no\_root\_squash and the observed file behavior:  
   In the NFS, there is a mount option called root\_squash, which maps any file operations initiated by the root user on the client machine (UID 0) to a non privileged user on the NFS server – typically and in our case nfsnobody. This mechanism is designed to prevent remote root users from having root-level access to files on the server.   
   When no\_root\_squash is enabled, this behavior is disabled. The root user on the client machine retains root privileges on the NFS share, allowing them to create and modify files as root on the server side.
2. Screenshots or logs showing the file creation process and permissions:

As part of the exercise, I tested how file ownership behaves under different NFS export configurations. In the screenshot, you can see the result of running ls -lah on the mounted directory after creating two test files: no\_squash and root\_squash.   
The file no\_squash was created while no\_root\_squash was enabled, and it’s owned by root, which shows that the root user’s permission from the client were preserved on the server.

The file root\_squash, on the other hand, was created with the default root\_squash setting, and it’s owned by nfsnobody, meaning root access was mapped to a non-privileged user.  
This test clearly demonstrates the difference between the two options and their effect on file permissions.



Section 3: Networking & Performance Analysis

1. Linux Client Performance Troubleshooting:  
   There are several main reasons that can cause performance issues on Linux. To diagnose the problem efficiently, I go through each cause, step by step, starting with the most common ones and those that are the easiest and fastest to check. This way, I can quickly find the root cause and focus on fixing it.  
   Also, if the issue happens repeatedly, I start by checking the last cause that triggered it in order to save time and focus on the most likely problem.

**Step 1 – Check CPU Load**I try to find if there are any processes consuming too much CPU or if the CPU itself is overloaded. This is important because if the CPU is busy, the whole system slows down, and it's usually one of the most common and easiest things to check first.  
To check this, I use top -o cpu which shows me a real-time view of processes sorted by CPU usage, and also ps aux --sort=-%cpu to quickly list the top CPU-consuming processes.  
For every process that uses a lot of CPU, I will write it down separately so I can investigate it further later.

4o

**Step 2 – Check Memory pressure**Next, I’ll check if the system is running low on memory. This could explain slow performance or unexpected behavior like processes getting killed.  
I use top -o %MEM to sort processes by memory usage. I also write down any process that uses a lot of memory so I can look into it later.  
To get a more detailed view, I use free -h to see the total memory, how much is used, and how much is available. If I see that swap is heavily used, that’s usually a sign the system ran out of RAM and started swapping to disk, which slows things down.  
I may also run ps aux --sort=-%mem to quickly see the top memory consumers.

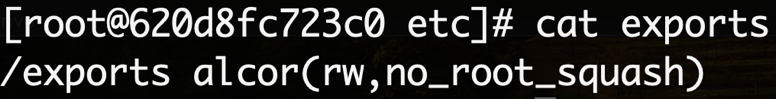
**Step 3 – Check system's disk I/O pressure**I want to understand if the system is spending a lot of time waiting on disk operations. If the disk is too slow or overloaded, even fast processes can get stuck waiting for data to be read or written, which makes the whole system feel slow. To check this, I use the iostat -xz 1 command. The -x flag gives me detailed stats per device, -z hides empty lines, and 1 updates the output every second. I mainly look at %util (how busy the disk is — close to 100% is bad) and await (how long processes wait for disk access).  
If I see high values, I try to figure out which processes are causing it using tools like iotop -o, which shows real-time disk activity per process, or pidstat -d 1, which reports per-process disk read/write rates. If a specific process is causing a lot of I/O, I’ll dig deeper into what it's doing and whether it’s expected — for example, log writing, backups, or maybe a bug causing excessive writes.

For the processes I identified as consuming high CPU or using a lot of memory, I will investigate them in depth in step 4 (using strace) and step 5 (checking the kernel stack) to understand the root cause of the issues and why they are causing the load.

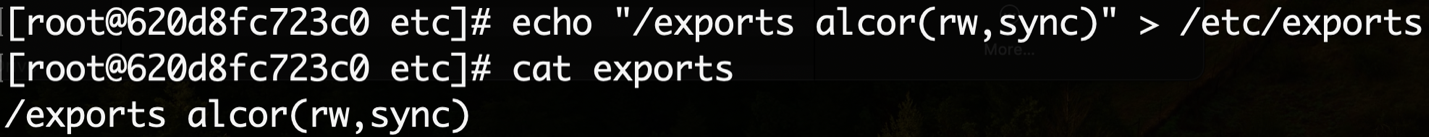
**Step 4 – Debug system call tracing**  
If a process’s behavior is unclear, I use strace -p <pid> or strace -c -p <pid> to capture the system calls it makes and how much time each takes.  
This helps me spot slow or failing system calls causing performance issues, such as long waits on files, sockets, or other resources. I can also save the trace into a file and investigate it later by using the flag -o .

**Step 5 – Inspect kernel stack**To understand what the kernel is doing on behalf of a stuck process, I look at /proc/<pid>/stack.  
This shows the kernel stack trace of the process’s thread, revealing if it’s blocked in kernel space (e.g., waiting for I/O or locks).  
It helps diagnose low-level issues not visible from userspace tools.

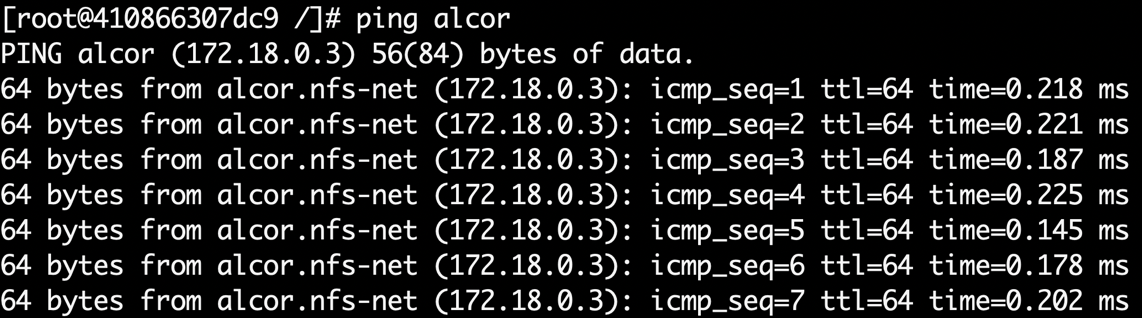
1. Networking Concepts:
2. MTU (Maximum Transmission Unit) defines the largest size of a packet that can be sent over a network interface without needing to be fragmented. It’s used to make sure data packets fit the limits of the network so they can be transmitted efficiently and without errors.
3. If packets are too big, they need to be split up, which adds overhead and can slow down communication.
4. The command sudo ss -i shows detailed info about active TCP connections, including internal parameters like retransmissions, congestion window, and packet sizes. This helps me understand how the TCP connection is behaving and if any limits like MTU are affecting performance. It also shows the state of the connections, for example whether it is listening or established (connected).   
   Additionally when using sudo, you get elevated priviliges allowing you to see all connections on the system, including those belonging to other users and system processes. Without sudo, a regular user only sees basic information, usually limited to their own connections.
5. Advmss is the maximum segment size that a TCP endpoint tells its peer it can receive in a single TCP segment, excluding headers. It directly affects how much data can be sent in each packet without fragmentation. If advmss is smaller than the MTU, the sender will send smaller packets, which can reduce fragmentation  
   but may affect throughput. You can see the advmss in the output of the sudo ss -I command.   
   The advmss is set during the TCP 3-way handshake. When a connection starts, each side tells the other what’s the biggest TCP segment it can receive. Advmss is what my machine offers to the other side.
6. PMTU (Path MTU Discovery) helps make sure that the packets sent over the network are not too big for any device along the way. It finds the smallest MTU (Maximum Transmission Unit) on the path and adjusts the packet size to fit, so they don’t get dropped.   
   For example when the NFS client tries to read or write a big file, it sends many packets to the server. If one of the routers in the path only allows smaller packets (for example, 1300 bytes), but the client sends bigger packets (like 1500 bytes), the packets might be blocked or lost. With PMTU, the system finds out that the biggest allowed packet is 1300 bytes and starts sending smaller packets. This helps the NFS connection work smoothly without losing data.
7. NFS Mount Options:
   1. Select three mount options and describe their purpose:
      1. RW (read-write) - Lets the client read and write files on the NFS share. In our exercise, I define it in the etc/exports/ file as you can see here:



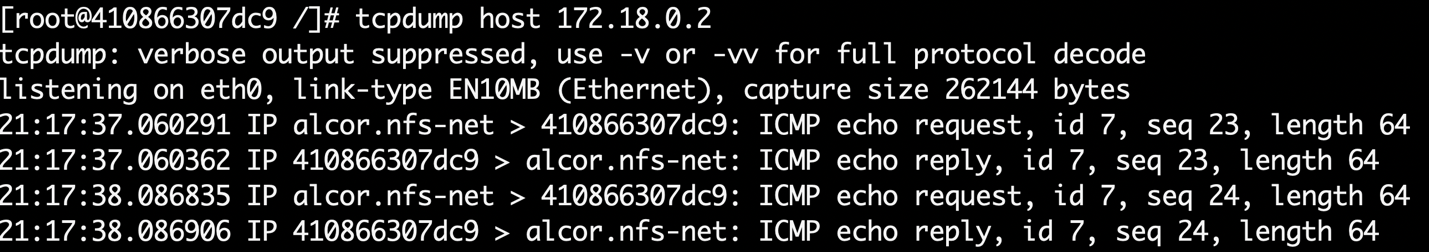
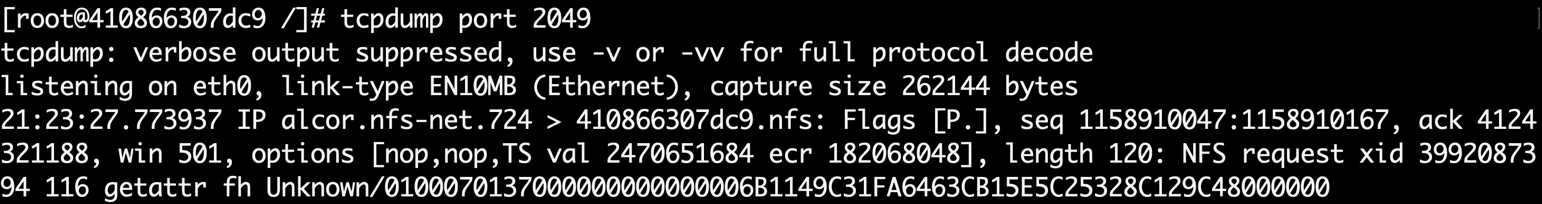
* + 1. Ro (read-only) - Lets the client only read files. They can’t make changes or delete anything. You can do it in the same way I showed on the first mount example and just change rw > ro.
    2. Sync - Makes sure that all data is written to disk immediately before confirming the write was successful.



* 1. Explain when and why you would use each mount option:
     1. rw (read-write) - Use this when the client needs to edit or add files on the shared folder. Otherwise the NFS server will deny access, and the client won’t be able to mount the shared folder.
     2. Ro (read-only) - Use this when you want to share files safely without anyone changing them.
     3. Sync - Use sync if you want to be sure your data is safe and won’t get lost. For example if you have a shared folder with important documents or databases, mounting it with sync helps avoid data loss even if the server suddenly crashes.

1. Network Traffic Analysis with tcpdump
   1. Capture and analyze network traffic using tcpdump:  
      I ran tcpdump, but no packets appeared because there was no network communication happening.  
      So I sent a ping from the server mizar to the client alcor, and that way I was able to see the packets.



* 1. Explain how to filter traffic for:
     1. A specific IP address
     2. A particular port (NFS traffic on port 2049)
     3. Only TCP or UDP packets

Tcpdump tcp

Tcpdump udp

* 1. How can tcpdump be used to troubleshoot NFS performance issues?  
     Because NFS is a network protocol, every time I use NFS it sends packets over the network. I can run tcpdump on the NFS server or client to capture and see these packets. This helps me check if the communication is working, if packets are delayed, dropped, or if there are errors. It’s a useful way to see if a performance problem comes from the network layer.
  2. Provide example commands and expected output interpretation:  
     tcpdump -i any port 2049 -nn  
       
     This captures all NFS traffic (which uses port 2049) on all interfaces.   
     The -nn option shows IP addresses and port numbers clearly, without trying to translate them.