

JOINT INSTITUTE FOR NUCLEAR RESEARCH

Meshcheryakov Laboratory of Information Technologies

**FINAL REPORT ON THE**

**START PROGRAM**

*Organization of database replication of a data storage system on magnetic tapes and plotting graphs in Grafana to monitor this system.*

**Supervisor:**

#### Dr. Alexander Moibenko

**Student:**

Polina Los, Kazakhstan,

AL-FARABI KAZAKH NATIONAL UNIVERSITY

**Participation period:**

July 29 – September 06,

Summer Session 2024

Dubna, 2024

**Annotation**

In this practice, the PostgreSQL database replication setup was performed, which is used in the magnetic tape data storage system using the master-slave configuration. The main goal of the work was to create a slave server, which will reproduce the changes occurring on the master server, which ensures fault tolerance of the system and the ability to restore data in case of failures.

The process of creating a replica included setting up access rights, configuring the necessary parameters on the master and slave nodes. Additionally, a mechanism for reading WAL (Write-Ahead Logging) directly from the master node to the slave was configured, which allows maintaining the relevance of data on the backup server.

After successful creation and configuration of replication, the data transfer and its relevance on the slave were checked. To analyze the volume of transferred information, data was collected from the slave server and graphs were built, displaying the activity of data transfer by each Mover and the overall activity. These graphs clearly demonstrate the efficiency of replication and allow to estimate the load on the system.

# Introduction

**The main problem**

Modern information systems require reliable data storage and protection mechanisms. One such system is Enstore, a tape storage system that uses PostgreSQL to record and store metadata and requires high availability and fault tolerance. In this context, database replication is an important means of ensuring reliable data storage and increasing system fault tolerance, as well as reducing the load on the main database server when executing user requests.

Data replication in PostgreSQL allows you to create copies of data on remote servers, which not only provides backup, but also minimizes system downtime in the event of a failure of the main server. To implement this approach, you need to choose the most suitable replication method, configure it, and ensure control over its operation. An important aspect is also real-time system monitoring for prompt identification and elimination of possible problems.

**History of the issue**

Database replication arose as a response to the need to ensure reliable data storage and minimize the risks associated with information loss. With the development of technology and the increase in data volumes, there was a need for more efficient and reliable replication methods. PostgreSQL has various approaches to replication, each of which has its own characteristics and areas of application.

Physical replication in PostgreSQL, implemented using WAL (Write-Ahead Logging), allows you to synchronize data between servers, ensuring their integrity and relevance. This method is developed and improved with each version of PostgreSQL, allowing you to solve problems of varying complexity - from simple backup to creating fault-tolerant clusters.

**Statement of the problem**

To ensure reliable operation of the magnetic tape storage system database, it was necessary to organize replication of the PostgreSQL 12 database, choosing the optimal method for this. It was also necessary to organize system monitoring using Grafana in order to track key performance indicators in real time and promptly respond to possible problems. Successful completion of these tasks will increase the fault tolerance of the system, ensure continuity of operation even in the event of failures on the main server, and reduce the load on the main database server during service read requests for monitoring the serviced system.

**Project objectives**

The main objectives of this work were:

1. An examination of the various PostgreSQL replication methods and the rationale for choosing the master-slave method.
2. Setting up and launching a slave server with subsequent testing of its operation.
3. Creating backups before and after setting up replication.
4. Using Grafana to visualize data from a PostgreSQL database.

# Scope of work

1. Study of different replication methods and their comparison.
2. Installing and configuring a slave server.
3. Creating configurations and starting the replication process.
4. Creating and testing backups before and after setting up replication.
5. Monitoring some metadata of the hosted system using Grafana, including creating dashboards and connecting PostgreSQL salve server as a data source.

***Types of replication***

There are different types of replication in database systems, including synchronous and asynchronous, as well as logical and streaming (physical). Synchronous replication provides high data consistency, since changes are recorded simultaneously on the primary and backup servers. However, it can slow down the system due to waiting for confirmation from the backup server. Asynchronous replication, on the other hand, provides high performance due to the fact that the primary server does not wait for confirmation from the replicas, but this can lead to temporary data inconsistency.

Logical replication copies data at the level of logical changes, such as SQL commands or table changes, allowing flexibility in choosing which data to replicate. Streaming replication, also known as physical replication, copies data at the database block level, making it more efficient and easier to configure, but less flexible than logical replication.

There are three main types of physical replication: block, file, and transaction. Block-level replication copies only changed blocks of data, which reduces the amount of data transferred and improves efficiency. However, it requires a complex mechanism to track changes at the block level. File-level replication is simpler to implement, but transfers entire files, which increases the amount of data and reduces efficiency with frequent changes. Transaction-level replication uses transaction logs, transferring only changes, which ensures data integrity, but requires support from the DBMS and is more difficult to manage.

Of these methods, I chose replication using WAL (Write-Ahead Logging), which is a transaction-level replication. This method provides high data integrity and allows only transactional changes to be transmitted, which reduces the amount of data and ensures efficient replication, especially in systems with high load and frequent changes.

### The process of creating replication and installing PostgreSQL 12 on the slave server

One of the key tasks for creating PostgreSQL database replication was the deployment of a slave server with PostgreSQL version 12. This stage was the beginning of the process of synchronizing two servers, where one of them acts as the main (master), and the second one acts as a replica (slave). The installation of PostgreSQL on the slave server had to match the features of the operating system (Scientific Linux 7), and this choice was determined by the capabilities of the package management system and supported repositories.

Several approaches were considered for installing PostgreSQL. The main emphasis was placed on installation via the dnf/yum package manager, as this method provides maximum ease of use and automatic dependency management. In addition, the system receives regular updates, including security patches, which is an important criterion in environments where it is necessary to ensure stable operation of databases. To obtain version PostgreSQL 12, a specialized PostgreSQL repository was used, which was connected to the system via the package:

sudo yum install -y postgresql12-server postgresql12

Alternative installation methods included building PostgreSQL from source and using PostgreSQL containers deployed via Docker. However, in this case, these options were less preferable due to their complexity or dependence on additional infrastructure elements. For example, installing from source provided finer customization but required more effort and time, while containerization added complexity in terms of data management.

### Rationale for choosing dnf/yum for installation

The choice of the dnf/yum package manager as the main installation tool was due to its simplicity and reliability. In the context of the Scientific Linux 7 operating system, compatible with Red Hat Enterprise Linux, using this package management system was a natural choice, as it integrates into the overall ecosystem, allowing for easy installation and updating of software products. This is especially important for the long-term maintenance of PostgreSQL security and functionality.

### PostgreSQL version clarification

For replication between master and slave to function correctly, the same version of PostgreSQL must be installed on both servers. The PostgreSQL version on the master server was confirmed with the commandsystemctl | grep postgreswhich showed that version 12.17 was being used:

postgresql-12.17.service - PostgreSQL 12.17 database server

Loaded: loaded (/usr/lib/systemd/system/postgresql-12.17.service; enabled; vendor preset: disabled)

Active: active (running) since Mon 2024-08-10 09:22:16 UTC; 2h 45min ago

Thus, during installation on the slave server, PostgreSQL version 12 was deployed in the same way via the package manager to avoid possible conflicts when setting up replication. The command used was:

yum install postgresql12 postgresql12-server

### Creating a replication

To ensure uninterrupted operation of the system, replication was configured based on streaming replication, which uses WAL (Write-Ahead Logging) logs. This is one of the most effective synchronization methods, as it ensures minimal delays in data transfer between servers.

A special user with replication rights was created on the master server.

To set up streaming replication, changes were made to the PostgreSQL configuration files on the master server, such as postgresql.conf and pg\_hba.conf. An important parameter in this process was the WAL log generation level, which was set to replica. In addition, a maximum limit on the number of processes sending WAL to the slave server was configured, and network addresses for listening for connections from the slave were defined.

Backup database

Before you start changing the database configuration, you should create a backup copy. This will help to avoid data loss in case of errors in replication setup. Backup can be performed using the pg\_basebackup command or using the pg\_dump utility.

An example command to create a backup using pg\_basebackup:

sudo -u postgres pg\_basebackup -D /path/to/backup -F t -z -P

This command will create a backup copy of the database and save it in the specified directory.

Setting up replication and restarting PostgreSQL

To apply new settings in PostgreSQL and set up replication between the master and slave servers, you need to perform several steps: backing up the database on the master, setting up configuration files, restarting the servers, and synchronizing data.

First, you need to log into the PostgreSQL database on the master server:

sudo -u postgres psql

Creating a user for replication on the master server: A user with replication rights is created on the master server:

CREATE USER replication REPLICATION CONNECTION LIMIT 2;

This setting specifies that the replication user can have a maximum of 2 simultaneous connections to the PostgreSQL server.

Changes in configuration files

On the master server:

1. File `postgresql.conf` (directory `/var/lib/pgsql/12/data).A line is added to the main configuration to tell PostgreSQL to use additional configurations:

include\_dir = 'conf.d'

1. File 1\_replicate.conf (directory /var/lib/pgsql/12/data/conf.d): This file contains the parameters necessary to configure replication:

listen\_addresses = 'localhost, 159.93.227.192'

wal\_level = replica

max\_wal\_senders = 10 - to configure the maximum number of processes sending WAL.

wal\_keep\_segments = 512—to keep the minimum number of WAL segments on disk.

3. File `pg\_hba.conf` (directory /var/lib/pgsql/12/data): To allow slave server connection::

host replication replication 159.93.227.193/32 trust

On the slave server:

1. File `postgresql.conf` (directory `/var/lib/pgsql/12/data`). As on the master server, a line is added to include a subdirectory with additional configurations::

include\_dir = 'conf.d'

1. File 1\_replicate.conf (directory /var/lib/pgsql/12/data/conf.d/): This file contains replication parameters for the slave server:

listen\_addresses = 'localhost, 159.93.227.193'

wal\_level = replica

max\_wal\_senders = 10

wal\_keep\_segments = 512

hot\_standby = on

1. File `pg\_hba.conf`(directory /var/lib/pgsql/12/data): To enable connection to the master server:

host replication replication 159.93.227.192/32 trust

3. File `2\_replica.conf` (directory `/var/lib/pgsql/12/data\_hot\_standby/conf.d/`):

To restore WAL logs from an archive:

restore\_command = 'unxz < "/diskb/enstore/backups/pg\_xlog\_archive/enstore/%f.xz" > ">p"'

Restart PostgreSQL to apply the new settings

1. Restart PostgreSQL on the master server (IP address: 159.93.227.192):

root# systemctl restart postgresql-12

2. Restart PostgreSQL on the slave server (IP address: 159.93.227.193):

- Stop the current PostgreSQL process:

root# systemctl stop postgresql-12

- Starting PostgreSQL after all configuration operations are completed:

root# systemctl start postgresql-12

Replicating data from master to slave

1. Stopping the database on the slave server:

root# systemctl stop postgresql-12

2. Deleting existing data on the slave server:

rm -rf /var/lib/pgsql/12/data/

3. Create a backup copy from master to slave:

The `pg\_basebackup` command is run on the slave server:

pg\_basebackup -h 159.93.227.192 -U replication -D /var/lib/pgsql/12/data -Fp -Xs -P -R

Command parameters:

- `-h 159.93.227.192` — IP address of the master server.

- `-U replication` — user for replication.

- `-D /var/lib/pgsql/12/data` — directory for storing data on the slave.

- `-Fp` — plain format.

- `-Xs` — copy WAL files.

- `-P` — display progress.

- `-R` — automatic creation of configuration for replication.

4. Launching the database on the slave server:

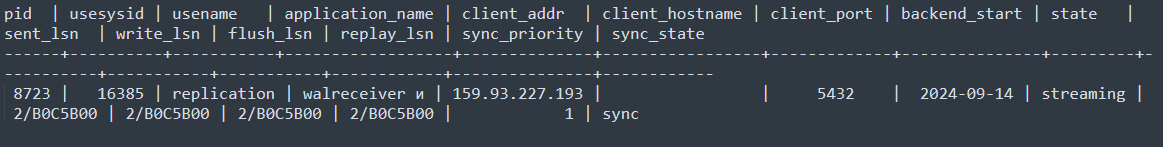
Once the setup and data copying is complete, the database server starts:

root# systemctl start postgresql-12

### These steps complete the replication setup, and the slave server will begin receiving data from the master server in real time. Setting up a hot standby on the slave server allows you to perform read requests on the replica server, expanding the ability to use the replica not only as a backup mechanism, but also for performing data read operations.

After completing the replication setup, it is necessary to make sure that the replication process is functioning successfully. For this purpose, a check was performed on both servers: master and slave.

On the master server, the replication status was checked using the pg\_stat\_replication system view, which displays active replication connections. The command used to check was SELECT \* FROM pg\_stat\_replication. The output showed information about the slave server, including the process identifier (pid), state (state - streaming), and WAL lag (parameters sent\_lsn, write\_lsn, flush\_lsn, replay\_lsn), confirming an active connection and correct data replication.

On the slave server, replication status was checked by analyzing PostgreSQL logs located in the /var/lib/pgsql/12/data/log directory. The log files contain records of receiving and applying WAL logs. An example of a log entry:

2024-09-14 14:23:15.123 UTC [23142] LOG: started streaming WAL from primary at 2/B0C5A98 on timeline 1

2024-09-14 14:23:15.456 UTC [23142] LOG: received WAL data up to 2/B0C5B00

2024-09-14 14:23:15.789 UTC [23142] LOG: applied WAL up to 2/B0C5B00

These records indicate that the slave server is successfully receiving and applying incoming replication data, confirming that the replication mechanism is working correctly.

In addition, a check was performed at the database level: changes made to the database on the master server appeared on the slave server with minimal delay, which indicated that the replication mechanism was working correctly.

This process ensures the creation of a reliable replica of the database that is synchronized with the primary server, providing data redundancy and high system availability.

### Plotting data volume and data transfer rate graphs in Grafana

This project involved developing and customizing a series of Grafana graphs to visualize data volumes and transfer rates in the Enstore storage system. Grafana, with its flexibility and powerful visualization features, has proven to be an excellent tool for creating informative and visual dashboards.

A dashboard is an interactive panel that combines various data visualizations, such as graphs, tables, and metrics, in one place. This allows you to conveniently monitor key indicators in real time and quickly respond to changes in the system. In our project, dashboards were used to monitor the volume and speed of data transfer.

The process of inviting someone to a shared Grafana workspace typically involves the following steps:

**Create an invitation**: The administrator or owner of the workspace creates an invitation link. This is done in the user management section, where you can select the "Invite" option.

**Setting up access rights**: When creating an invitation, the administrator can set the access level for the new user, such as view only or the ability to edit dashboards.

**Authorization**: The user may be invited to log in to their Grafana account or create a new one if they don't have one.

**Access to dashboards**: After successful login, the user gets access to dashboards and other workspace resources, depending on the access rights set.

Let's move on to creating a new dashboard.

1. Setting up a connection and creating a data source

First, you need to establish a connection to the PostgreSQL data source in Grafana. This is important, as setting up the connection correctly ensures that the data is received correctly and can be visualized. The process of establishing a connection includes the following steps:

* Adding a new data source: In Grafana, this is done via the "Configuration" menu and "Data Sources". Here, PostgreSQL is selected as the data source type.
* Filling in connection parameters: It is important to specify the parameters correctly, such as:
  + Host (in our case, the slave server address is 159.93.227.193/32)
  + Port (default is 5432)
  + Database name
  + Username and Password

For security, you should use TLS if your network security settings allow it. TLS (Transport Layer Security) is a cryptographic protocol that provides a secure connection between a client and a server. It encrypts data transmitted over the network to prevent interception and unauthorized access. Using TLS helps ensure the confidentiality and integrity of data, as well as authentication of the parties, which is especially important when transmitting sensitive information such as credentials and transaction data.

* Checking the connection: After entering all the required details, it is recommended to perform a connection test to ensure that Grafana can successfully communicate with PostgreSQL.

### 2. Creating dashboards and customizing them

Creating dashboards in Grafana allows you to combine graphs on one screen, which greatly simplifies monitoring and data analysis. To create a dashboard in Grafana, you need to open the Grafana interface and select the option to create a new dashboard from the main menu. Then add a panel by selecting the type of visualization and configure its parameters. After that, save the dashboard, giving it a name and description. The configured dashboard can be edited and updated as needed, adding new panels and configuring them to display relevant data.

3. Creating variables and plotting graphs

Once the connection was set up, the next step was to create variables that would allow the data displayed on the charts to be dynamically changed. In this case, a variable that included all movers was created to allow the selection of a specific mover or group of movers in the dashboard.

Algorithm for creating a variable

On the main page of the created dashboard, you need to click on the dashboard icon or select "dashboard settings". Then select the "variables" tab and click "add variable". Select the variable name, in our case "mover", and the variable setting type "query" with the PostgreSQL data source. Generate a query:

`SELECT DISTINCT mover FROM xfer\_by\_day\_by\_mover ORDER BY mover;`.

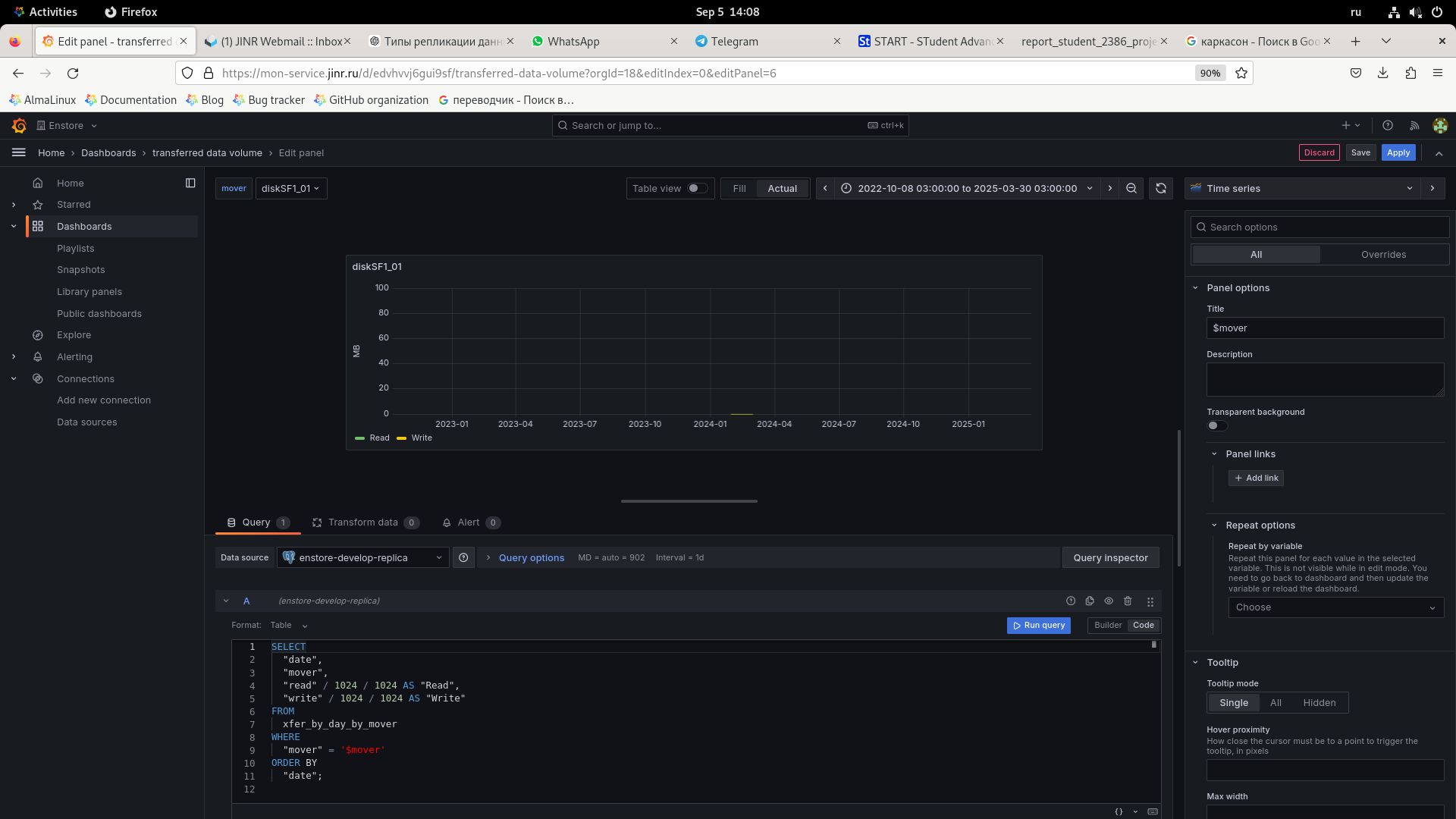
In the "Label" section, specify how the variable will be displayed in the dashboard, for example, "Select mover". Click "Update" to save the variable settings. Now that the variable is created, you can use it in chart queries. For example, in a SQL query for charts, you can use:

`SELECT \* FROM your\_table WHERE mover IN ($mover);`

This will allow you to dynamically change the data in the graphs depending on the selected mover.

4. Data Transfer Volume Graphs

To plot the data transfer volume graphs, the following SQL query was used:



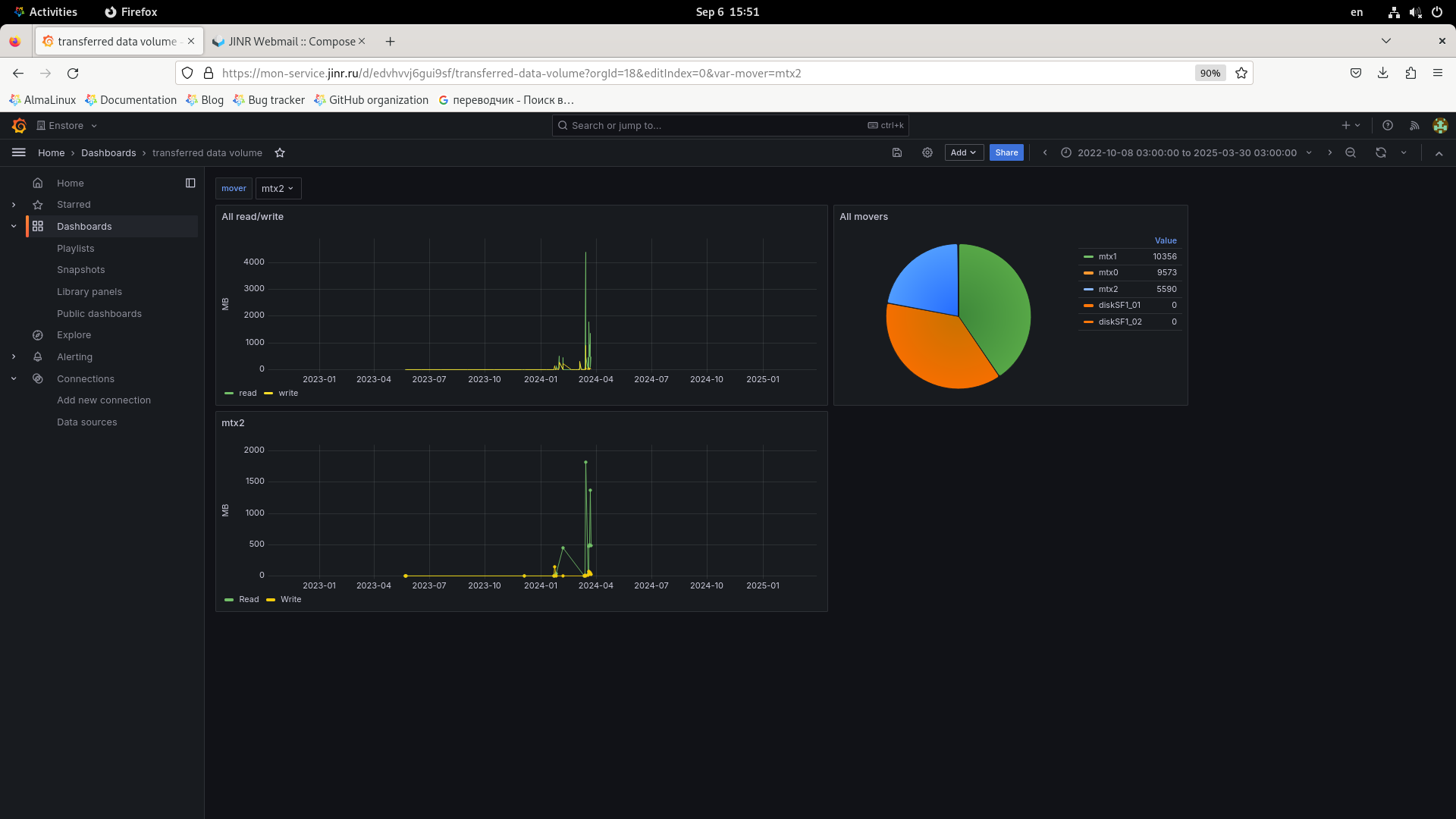
This query allows you to extract data on the read and write volume for a specific mover, converting it from bytes to megabytes. The results are displayed as a time series, which allows you to analyze the dynamics of the data transfer volume both in general and for each individual mover.

5. Data transfer rate graphs

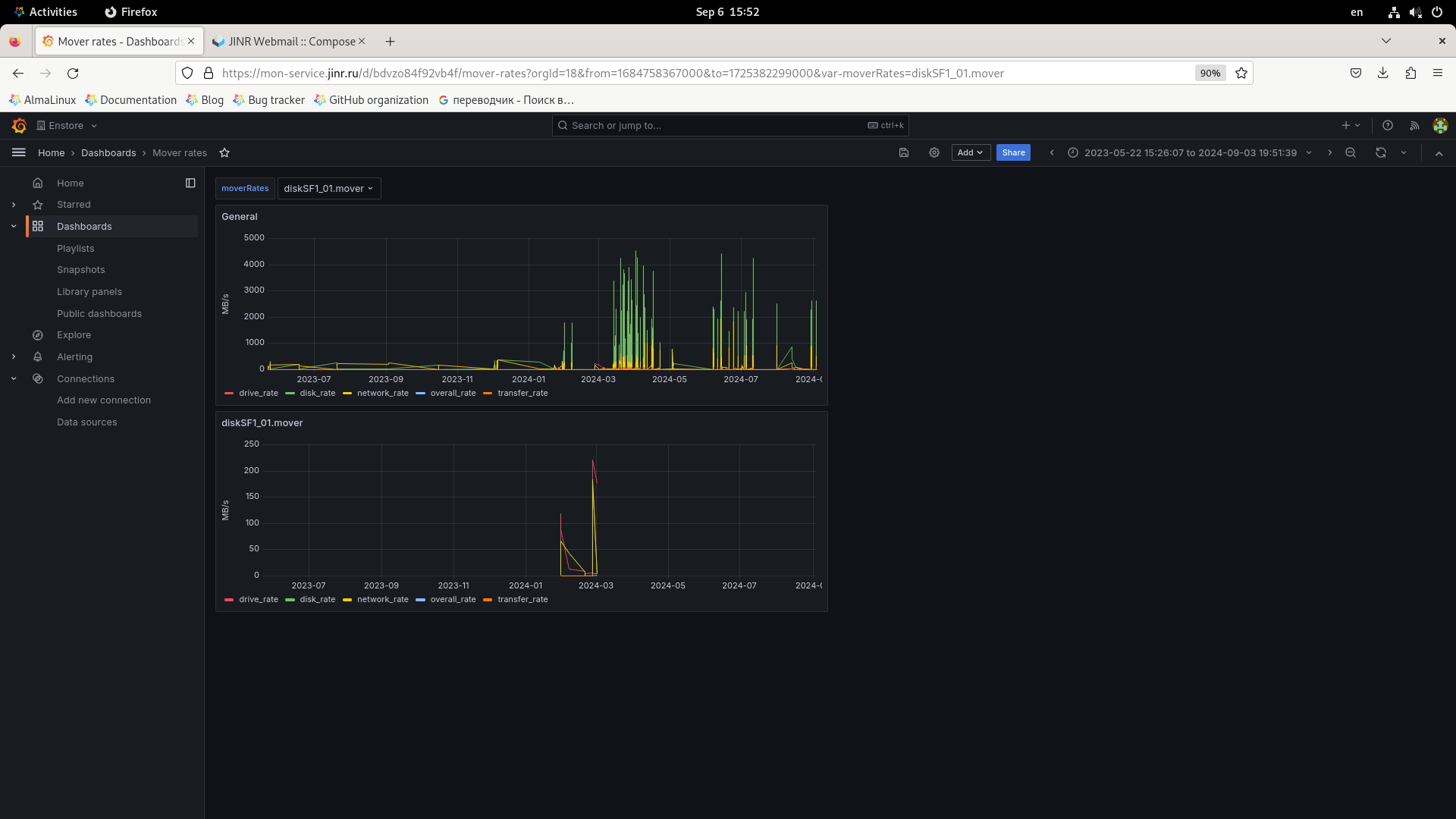
The data rate graphs were constructed using the same approach as the data volume graphs. Using time series makes it easy to track changes in data rate and identify potential problems or trends.

Below are examples of graphs created as part of this project:

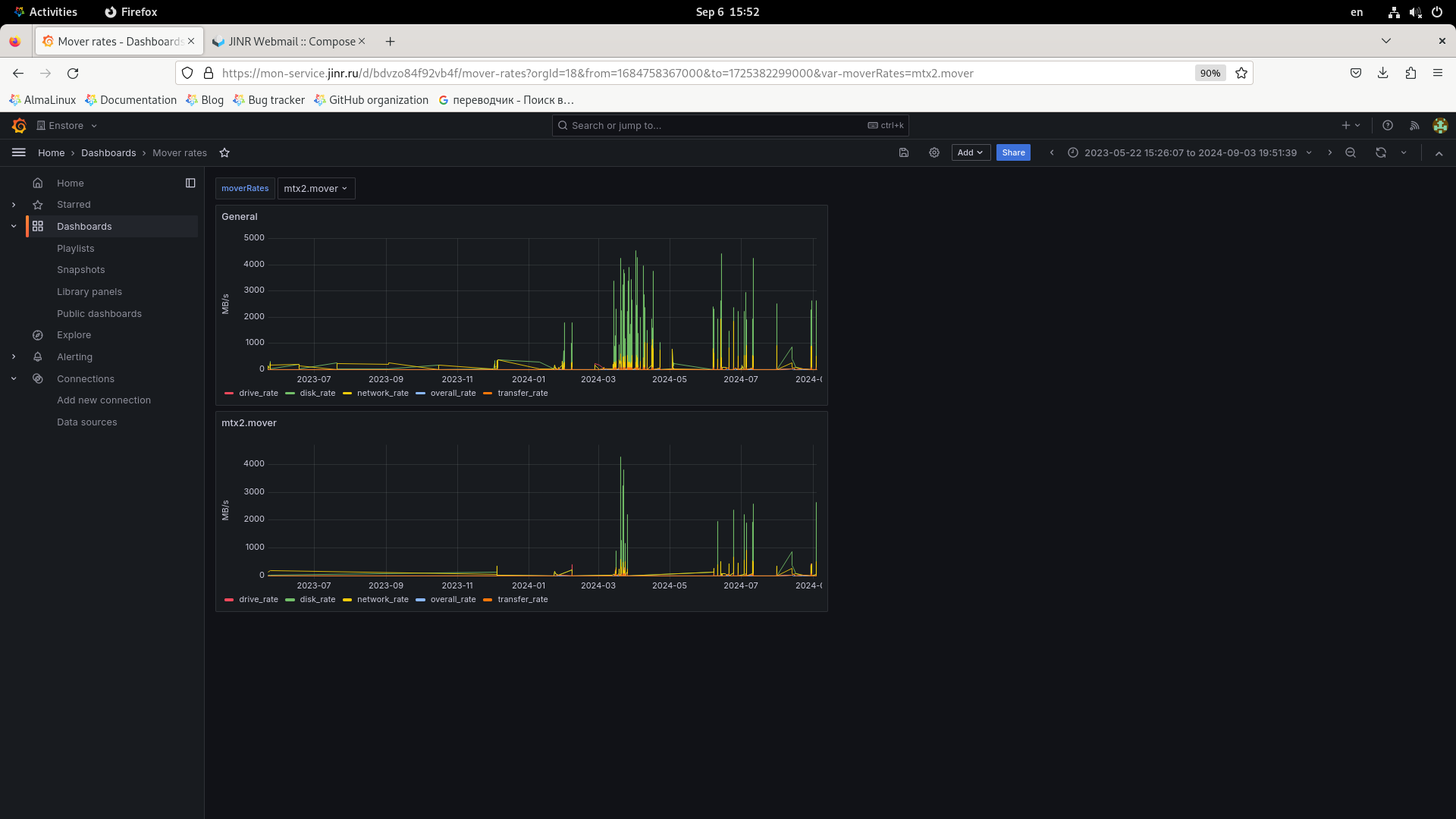
Total data transfer volume graph:

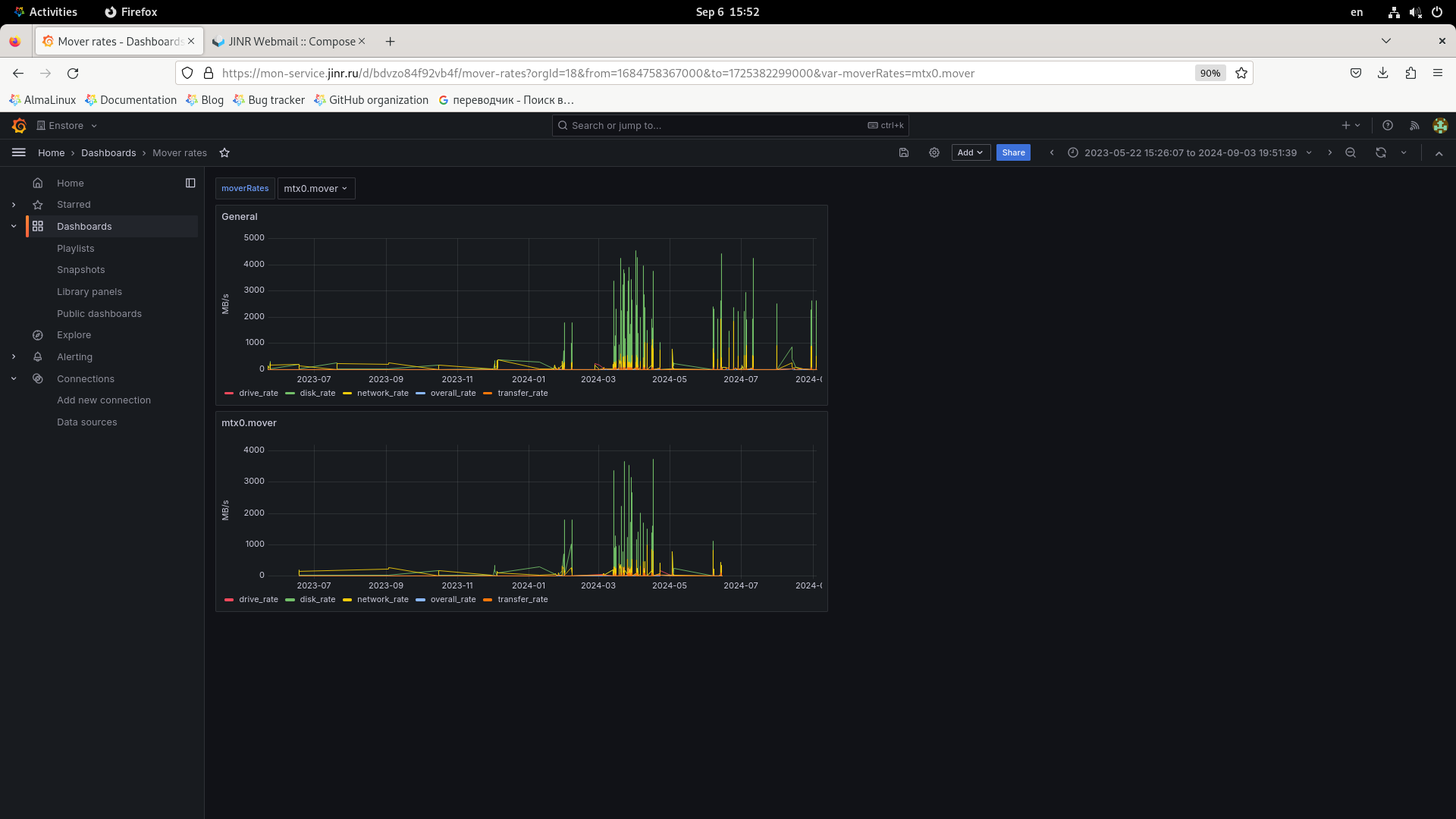


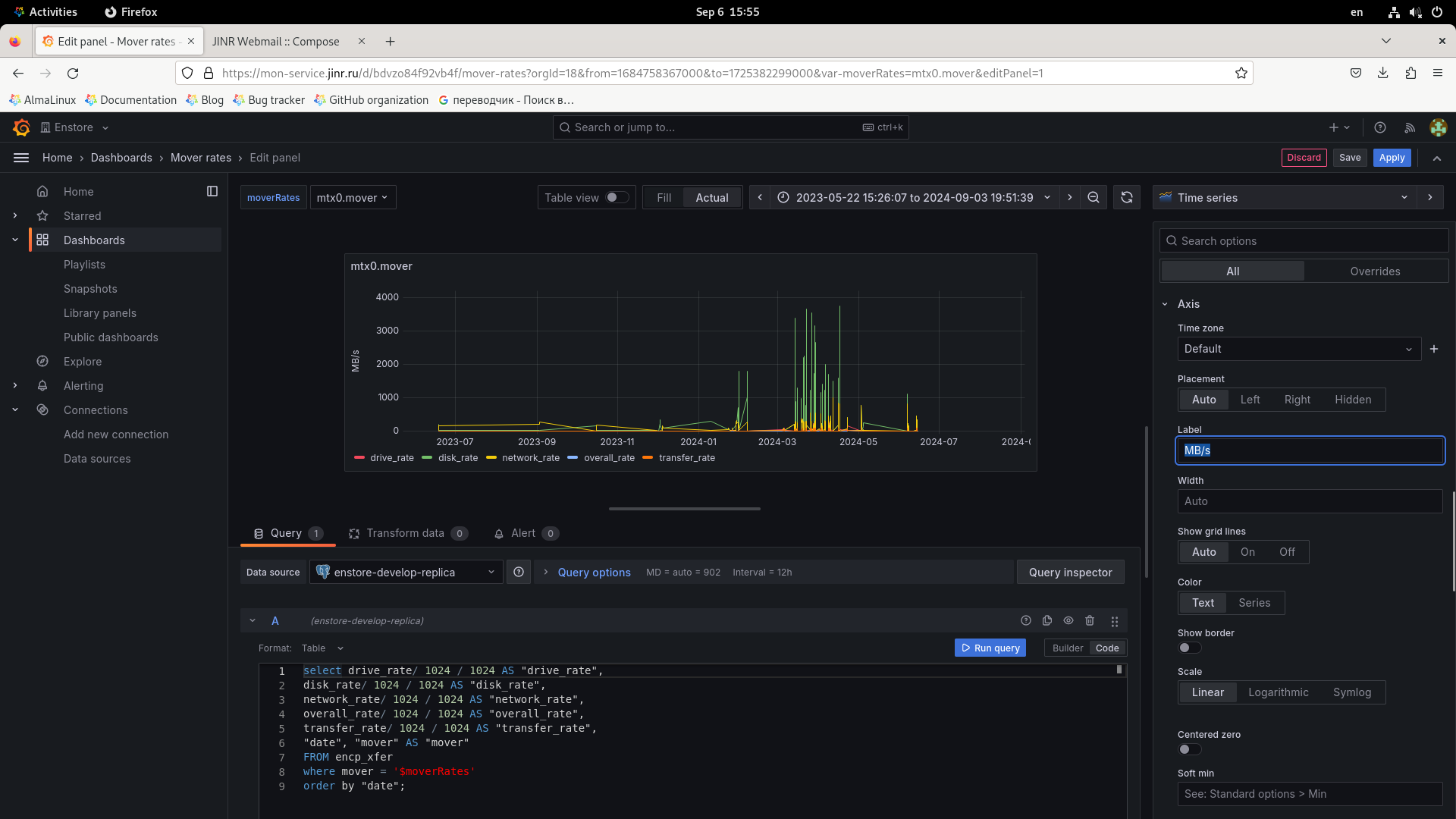
Data transfer rate graph:



Data transfer schedule for individual movers:







#### Benefits of Using Grafana

Grafana offers a number of significant advantages that make it an ideal tool for data visualization:

* Interactive dashboards: Grafana allows you to create dynamic and interactive dashboards that provide easy access to various metrics and data.
* Support for multiple data sources: Grafana can connect to a variety of data sources, including SQL and NoSQL databases, allowing it to be integrated with a variety of data storage systems.
* Flexibility of customization: The platform offers a wide range of settings for charts and dashboards, allowing you to create customized visualizations to suit your specific requirements.

In conclusion, creating graphs in Grafana to monitor the volume and speed of data transfer in the Enstore system has shown the high efficiency of this tool. Thanks to Grafana, it was possible to create convenient and informative dashboards that significantly simplify the analysis and monitoring of data.

### Conclusion

Internship based on Enstore DB Hot Standby and Grafana monitoring system gave an opportunity to delve into modern technologies of data management, monitoring and their safe storage. Setting up PostgreSQL replication, WAL archiving and monitoring of data transfer speed and volume allowed to gain valuable practical experience of working with highly loaded systems. This internship not only helped to master specific technical skills, but also expanded the understanding of the importance of building fault-tolerant systems.

Prospects for further development may include:

* Automation of data analysis processes to improve the speed of decision making;
* Implement more sophisticated replication and backup systems to improve security and performance;
* Using machine learning to predict potential failures and optimize storage resource usage.

### Sources for study

### PostgreSQL Documentation - The official PostgreSQL guide contains detailed information on setting up replication and restoring from backups.

1. **Grafana Documentation**– This resource will help you to gain a deeper understanding of working with dashboards and integration with various data sources.

Link: https://grafana.com/docs/

1. **Database Administration Books**– Books such as "PostgreSQL: Up & Running" and "The Art of PostgreSQL" can be useful for further delving into the topic.
2. **Stack Overflow Community**– An online forum where you can find answers to specific technical questions.

### Acknowledgments

I would like to express my deep gratitude to the START program and JINR for the opportunity to undergo such an important internship. I would like to express special gratitude to my scientific supervisor, Alexander Nikolaevich Moibenko, for his professional guidance, support and advice during the work process. His knowledge and experience helped me master aspects of working with databases and monitoring systems, as well as to better understand the importance of attention to detail when working with highly loaded systems.

I would also like to mention the friendly and highly professional team that supported me throughout my internship. Interaction with such experienced specialists gave me a unique opportunity to exchange knowledge and experience, which significantly enriched my professional skills.