Настраиваем асинхронную репликацию.

Pgmaster

# create replication user

CREATE ROLE replicator with login replication password 'reppassword'

# allow users to connect

echo 'host    replication         replicator     172.0.0.0/8    md5' >> /var/lib/postgresql/data/pg\_hba.conf

# Update pg.conf

echo 'ssl = off' >> /var/lib/postgresql/data/postgresql.conf

echo 'wal\_level = replica' >> /var/lib/postgresql/data/postgresql.conf

echo 'max\_wal\_senders = 4' >> /var/lib/postgresql/data/postgresql.conf

# restart postgres

pg\_ctl reload

Pgslave

#!/bin/bash

touch standby.signal

# Update pg.conf

echo "primary\_conninfo = 'host=pgmaster port=5432 user=replicator password=reppassword application\_name=pgslave'" >> /var/lib/postgresql/data/postgresql.conf

Имеем следующие строки подключения

"DatabaseSettings": {

"ConnStr": "Host=127.0.0.1;Port=15432;Username=dbuser;Password=dbpassword;Database=otusdb;Pooling=true;",

"ConnStrReplica": "Host=127.0.0.1;Port=25432;Username=dbuser;Password=dbpassword;Database=otusdb;Pooling=true;"

}

Запросы на чтение (/user/get/{id} и /user/search из спецификации) и переносим их на чтение со слейва

public async Task<(bool isSuccess, string msg, UserEntity user)> GetUserAsync(string id)

{

await using var con = await dbReplica.OpenConnectionAsync();

var sql = "SELECT id, first\_name, second\_name, sex, age, city, biography\r\nFROM public.\"user\"\r\n WHERE id = @id LIMIT 1;";

var items = con.Query<UserEntity>(sql, new { id = id });

if (items.Count() > 0) { return (true, "OK", items.First()); }

return (false, "Not found", null);

}

public async Task<(bool isSuccess, string msg, List<UserEntity> users)> SearchUserAsync(string firstName, string lastName)

{

await using var con = await dbReplica.OpenConnectionAsync();

var sql = "SELECT id, first\_name, second\_name, sex, age, city, biography\r\nFROM public.\"user\"\r\n";

var sqlConditions = new List<string>();

IEnumerable<UserEntity> items;

if (!string.IsNullOrEmpty(firstName) && !string.IsNullOrEmpty(lastName))

{

sql += "WHERE first\_name LIKE @firstname AND second\_name LIKE @secondname ORDER BY id;";

items = con.Query<UserEntity>(sql, new

{

@firstname = $"{firstName}%",

@secondname = $"{lastName}%",

});

}

else if (!string.IsNullOrEmpty(firstName))

{

sql += "WHERE first\_name LIKE @firstname ORDER BY id;";

items = con.Query<UserEntity>(sql, new

{

@firstname = $"{firstName}%"

});

}

else if (!string.IsNullOrEmpty(lastName))

{

sql += "WHERE second\_name LIKE @secondname ORDER BY id;";

items = con.Query<UserEntity>(sql, new

{

@secondname = $"{lastName}%"

});

}

else

{

sql += " ORDER BY id LIMIT 100;";

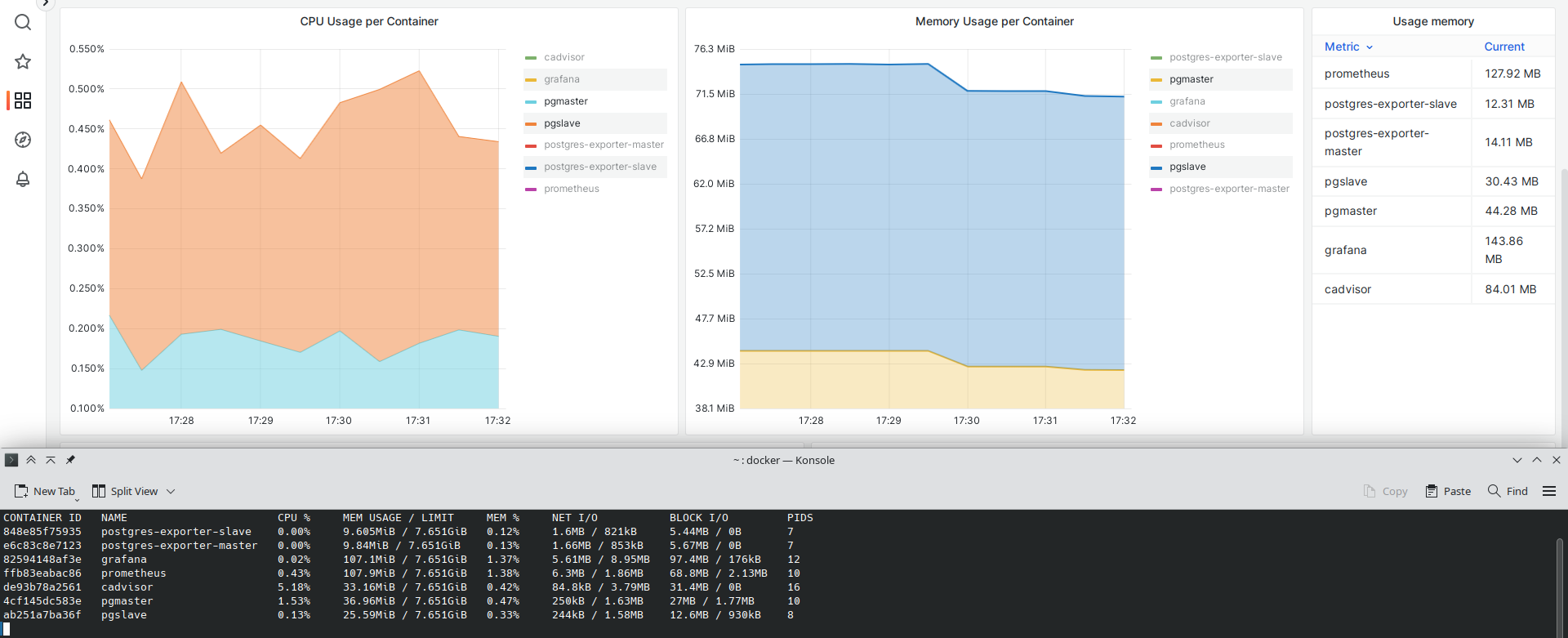
items = con.Query<UserEntity>(sql);

}

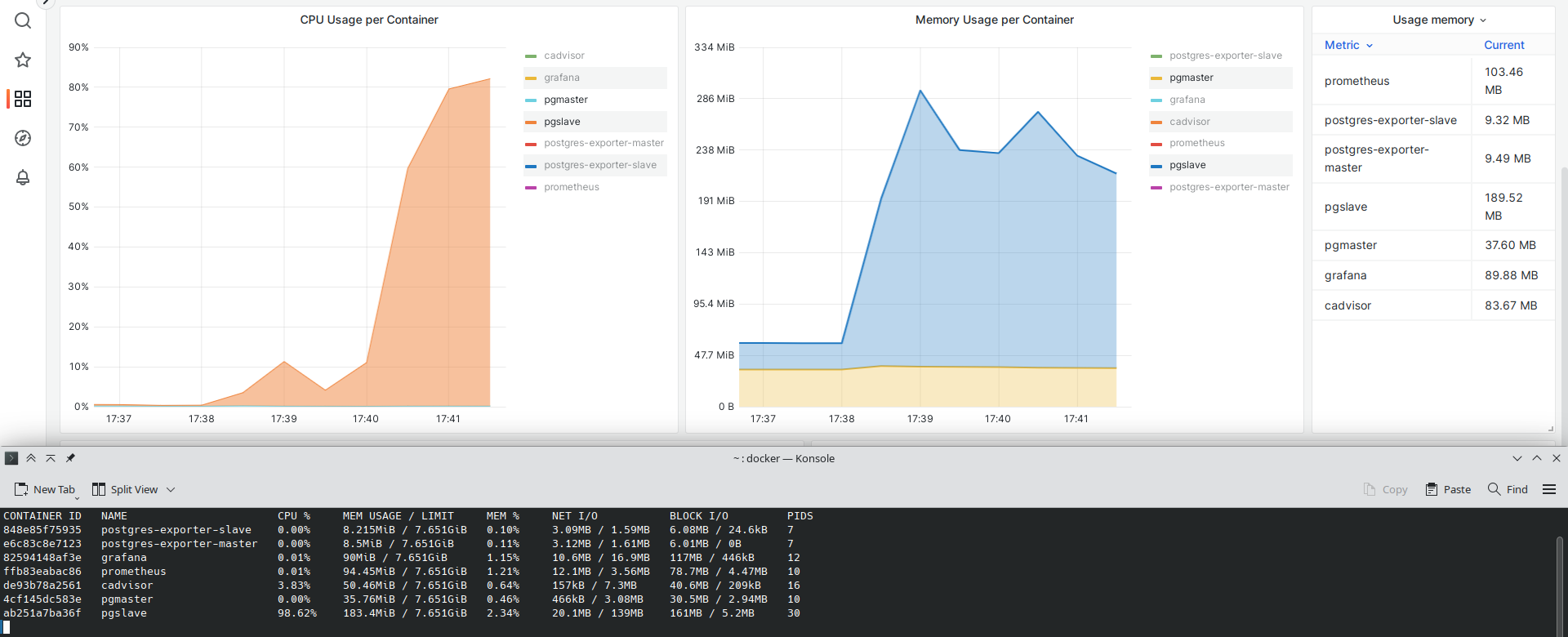
return (true, "OK", items.ToList());

}

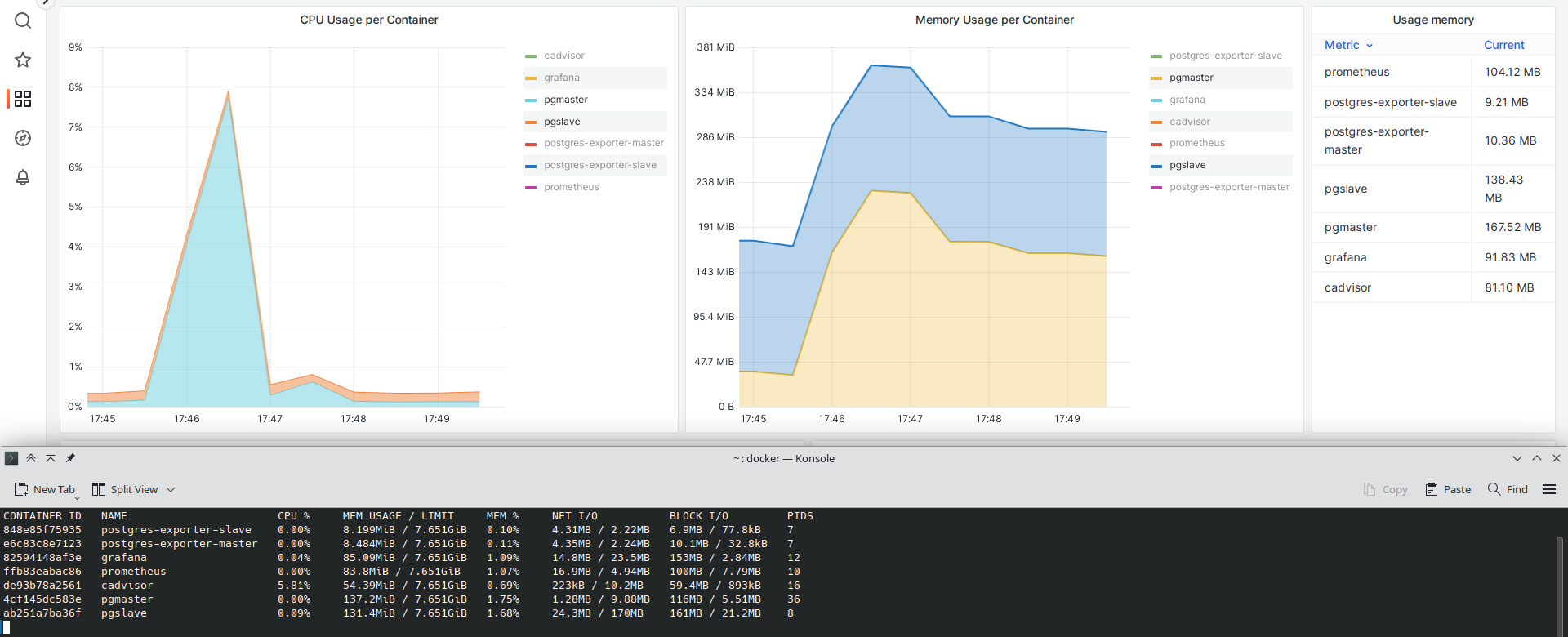
До подачи нагрузки



После подачи нагрузки (чтение с pgslave)



После подачи нагрузки (чтение с pgmaster)



Настраиваем кворумную синхронную репликацию

Создаем таблицу newtable для записи.

public async Task<(bool isSuccess, string msg, string userId)> AddNewTableRecordAsync(NewTableEntity user)

{

// Check connection to master

try

{

await using var con = await db.OpenConnectionAsync();

await using var cmdAccount = new NpgsqlCommand("INSERT INTO public.newtable\r\n(person, age, city)\r\nVALUES(@person, @age, @city);\r\n", con)

{

Parameters = {

new("person", user.Person),

new("age", user.Age),

new("city", user.City)

}

};

await cmdAccount.ExecuteNonQueryAsync();

return (true, "OK", user.Person);

} catch (Exception ex)

{

await using var con = await dbReplica.OpenConnectionAsync();

await using var cmdAccount = new NpgsqlCommand("INSERT INTO public.newtable\r\n(person, age, city)\r\nVALUES(@person, @age, @city);\r\n", con)

{

Parameters = {

new("person", user.Person),

new("age", user.Age),

new("city", user.City)

}

};

await cmdAccount.ExecuteNonQueryAsync();

return (true, "OK", user.Person);

}

}

Настройки для Pgmaster

# Делаем бекап для реплик

docker exec -it pgmaster pg\_basebackup -h pgmaster -D /backups -U replicator -v -P --wal-method=stream --checkpoint=fast

# Update pg.conf

synchronous\_commit = on

synchronous\_standby\_names = 'FIRST 1 (pgslave, pgslavesecond)'

Настройки для Pgslave и Pgslavesecondary

# Копируем бекап для реплик

sudo cp -r ./pgmaster-backups/\* ./pgslave-data

sudo cp -r ./pgmaster-backups/\* ./pgslave-second-data

# Создадим файл для реплик

touch pgslave-data/standby.signal

touch pgslave-second-data/standby.signal

# Меняем postgresql.conf на репликах

primary\_conninfo = 'host=pgmaster port=5432 user=replicator password=reppassword application\_name=pgslave'

primary\_conninfo = 'host=pgmaster port=5432 user=replicator password=reppassword application\_name=pgslavesecond'

# По итогу имеем для pgmaster

postgres=# select application\_name, sync\_state from pg\_stat\_replication;

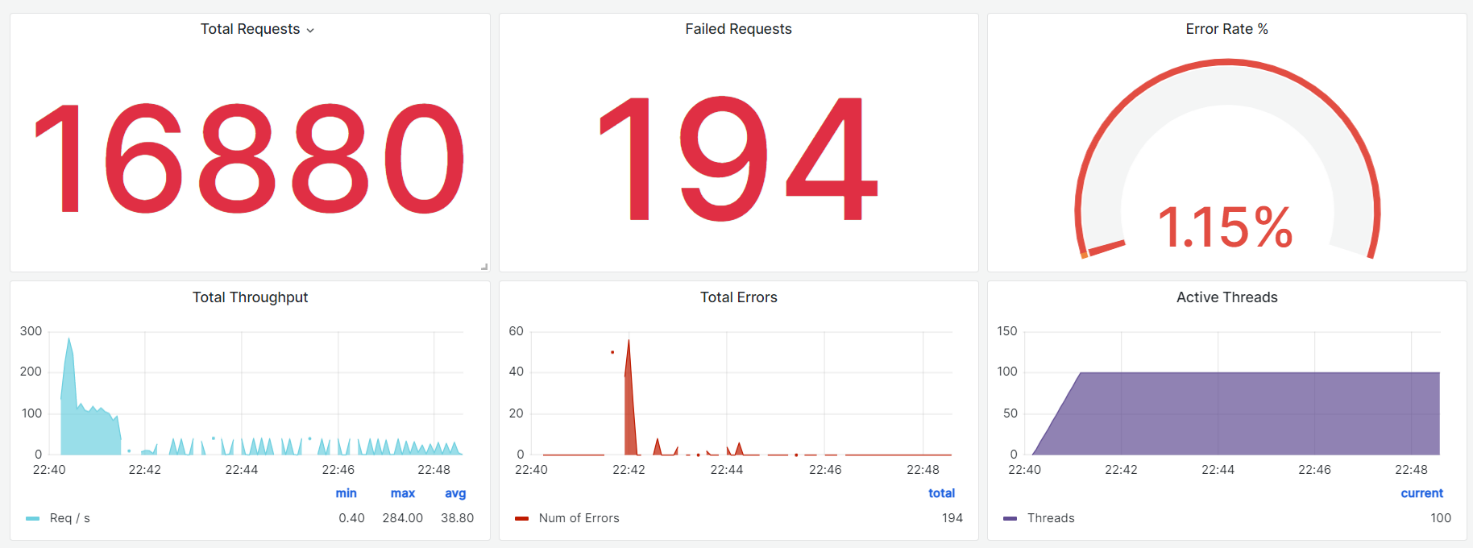
application\_name | sync\_state

------------------+------------

pgslave | sync

pgslavesecond | potential

Запускаем нагрузку на запись



На начало нагрузки в таблице newtable было 1000004 записи.

В определенный момент (22:42) делаем docker stop pgmaster.

В этот момент в таблице newtable 1005266 записи.

Для pgslave делаем

# Files

rm standby.signal

# psql

select \* from pg\_promote();

# Update pg.conf в расчете на то, что может поднимем pgmaster

~~primary\_conninfo = 'host=pgmaster port=5432 user=replicator password=reppassword application\_name=pgslave'~~

synchronous\_commit = on

synchronous\_standby\_names = 'ANY 1 (pgmaster, pgslavesecond)'

Для pgslavesecond делаем

# Update pg.conf

primary\_conninfo = 'host=pgslave port=5432 user=replicator password=reppassword application\_name=pgasyncslave'

По итогу имеем для pgslave

postgres=# select application\_name, sync\_state from pg\_stat\_replication;

application\_name | sync\_state

------------------+------------

pgasyncslave | quorum

Нагрузка не останавливалась и по итогу имеем записей в таблице newtable

Pgmaster: 1005266

Pgslave: 1008664

Pgslavesecondary: 1008664

