Known issues with upgrades to your instance

In this article

See an overview of workarounds for issues that impact the upgrade process for GitHub Enterprise Server, or impact your instance after you complete an upgrade.

About known issues with GitHub Enterprise Server upgrades

GitHub is aware of the following issues that could impact upgrades to new releases of GitHub Enterprise Server. For more information, see "Known issues" in the GitHub Enterprise Server release notes.

GitHub strongly recommends regular backups of your instance's configuration and data. Before you proceed with any upgrade, back up your instance, then validate the backup in a staging environment. For more information, see "Configuring backups on your instance" and "Setting up a staging instance."

Increased I/O utilization from MySQL 8 upgrade in GitHub Enterprise Server 3.10

If you upgrade from GitHub Enterprise Server 3.7 or 3.8 to 3.9 or later, an upgrade to the database software on your instance will increase I/O utilization. In some cases, this may affect your instance's performance.

GitHub Enterprise Server includes a MySQL database server supported by the InnoDB storage engine. GitHub Enterprise Server 3.8 and earlier use MySQL 5.7. In October 2023, Oracle will end extended support for MySQL 5.7. For more information, see Oracle Lifetime Support Policy on the Oracle Support website.

To future-proof GitHub Enterprise Server and provide the latest security updates, bug fixes, and performance improvements, GitHub Enterprise Server 3.9 and later use MySQL 8.0. MySQL 8.0 achieves a higher number of queries per second (QPS) due to a redesigned REDO log. For more information, see MySQL Performance: 8.0 re-designed REDO log & ReadWrite Workloads Scalability on DimitriK's (dim) Weblog.

After the upgrade to GitHub Enterprise Server 3.9, if you experience unacceptable degradation in the performance of your instance, you can collect data from your instance's monitor dashboard to confirm the impact. You can attempt to mitigate the issue, and you can provide the data to GitHub Support to help profile and communicate the real-world impact of this change.

Warning: Due to the nature of this upgrade, back up your instance's configuration and data before proceeding. Validate the backup in a staging environment. For more information, see "Configuring backups on your instance" and "Setting up a staging instance."

Collecting baseline I/O utilization data before the MySQL upgrade

Collect the baseline data before upgrading to GitHub Enterprise Server 3.9 or later. To collect baseline data, GitHub recommends that you set up a staging instance of GitHub Enterprise Server running 3.7 or 3.8 and restore data from your production instance using GitHub Enterprise Server Backup Utilities. For more information, see "Setting up a staging instance" and "Configuring backups on your instance."

You may not be able to simulate the load that your instance experiences in a production environment. However, it's useful if you can collect baseline data while simulating patterns of usage from your production environment on the staging instance.

1. Browse to your instance's monitor dashboard. For more information, see "Accessing the monitor dashboard."
2. From the monitor dashboard, monitor relevant graphs.
    - Under "Processes", monitor the graphs for "I/O operations (Read IOPS)" and "I/O operations (Write IOPS)", filtering for `mysqld`. These graphs display I/O operations for all of the node's services.
    - Under "Storage", monitor the graph for "Disk utilization (Data Device DEVICE-ID)". This graph displays the amount of time spent on all of the node's I/O operations.

Reviewing I/O utilization data after the MySQL upgrade

After the upgrade to GitHub Enterprise Server 3.9, review the instance's I/O utilization. GitHub recommends that you upgrade a staging instance of GitHub Enterprise Server running 3.7 or 3.8 that includes restored data from your production instance, or that you restore data from your production instance to a new staging instance running 3.9. For more information, see "Setting up a staging instance" and "Configuring backups on your instance."

1. Browse to your instance's monitor dashboard. For more information, see "Accessing the monitor dashboard."
2. From the monitor dashboard, monitor relevant graphs.
    - Under "Processes", monitor the graphs for "I/O operations (Read IOPS)" and "I/O operations (Write IOPS)", filtering for `mysqld`. These graphs display I/O operations for all of the node's services.
    - Under "Storage", monitor the graphs for "Disk utilization (Data Device DEVICE ID)" and "Disk Latency (Data Device DEVICE-ID)". These graph display the amount of time spent on all of the node's I/O operations, as well as overall disk latency.
        - Significant increases to disk latency could indicate that your instance is forcing disk IOPS to wait to complete.
        - You can corroborate an observation of increased latency by reviewing the graph for "Disk pending operations (Data Device DEVICE-ID)", which could indicate that the disk cannot sufficiently address all operations.

Mitigating impact of the MySQL upgrade

To address unacceptable degradation of performance, GitHub recommends the following solutions.

Before you test any mitigation procedure in a production environment, back up your instance, validate the backup, then test the procedure in a staging environment. For more information, see "Configuring backups on your instance" and "Setting up a staging instance."

- Adjust InnoDB's flushing method
- Upgrade your instance's storage

Adjust InnoDB's flushing method

To attempt to mitigate the performance impact, you can adjust InnoDB's flushing method to skip the `fsync()` system call after each write operation. For more information, see `innodb_flush_method` in the MySQL 8.0 Reference Manual.

The following instructions are only intended for GitHub Enterprise Server 3.9 and later.

Warning: Adjustment of the flushing method requires that your instance's storage device has a battery-backed cache. If the device's cache is not battery-backed, you risk data loss.

- If you host your instance using a virtualization hypervisor within an on-premises datacenter, review your storage specifications to confirm.
- If you host your instance in a public cloud service, consult your provider's documentation or support team to confirm.

1. SSH into your GitHub Enterprise Server instance. If your instance comprises multiple nodes, for example if high availability or geo-replication are configured, SSH into the primary node. If you use a cluster, you can SSH into any node. For more information about SSH access, see "Accessing the administrative shell (SSH)."

   ```
   ssh -p 122 admin@HOSTNAME
   ```

2. To validate the current flushing method for InnoDB, run the following command.

   Shell

   ```
   ghe-config mysql.innodb-flush-no-fsync
   ```

   By default, the command returns `false`, indicating that your instance performs an `fsync()` system call after each write operation.

3. To configure InnoDB to skip the `fsync()` system call after each write operation, run the following command.

   Shell

   ```
   ghe-config mysql.innodb-flush-no-fsync true
   ```

4. To apply the configuration, run the following command.

   Note: During a configuration run, services on your GitHub Enterprise Server instance may restart, which can cause brief downtime for users.

Shell

```
ghe-config-apply
```

5. Wait for the configuration run to complete.

Upgrade your instance's storage

You can reduce pending operations, increase IOPS, and improve performance by provisioning faster storage for your instance's nodes. To upgrade your instance's storage, back up your instance and restore the backup to a new replacement instance. For more information, see "Configuring backups on your instance."

Sharing data with GitHub

Finally, if you're willing to help GitHub understand the real-world impact of the upgrade to MySQL 8, you can provide the data you've collected to GitHub Support. Provide the baseline and post-upgrade observations from the monitor dashboard, along with a support bundle that covers the period when you collected data. For more information, see "About GitHub Support" and "Providing data to GitHub Support."

The data you submit helps GitHub continue to provide a performant product, but GitHub does not guarantee any additional mitigation steps or changes to the product as a result of the data you provide.

MySQL does not start after upgrade to GitHub Enterprise Server 3.9 or 3.10

During an upgrade to GitHub Enterprise Server 3.9 (from 3.7 or 3.8) or 3.10 (from 3.8 only), if MySQL did not gracefully shut down during the shutdown of the GitHub Enterprise Server 3.7 or 3.8 instance, MySQL will attempt to go through crash recovery when the GitHub Enterprise Server 3.9 or 3.10 instance starts up. Since GitHub Enterprise Server 3.7 and 3.8 uses MySQL 5.7 and GitHub Enterprise Server 3.9 and 3.10 have been upgraded to MySQL 8.0, MySQL will not be able to complete crash recovery.

If you are upgrading from GitHub Enterprise Server 3.9 to 3.10 then you will not be affected by this issue, as MySQL has already been upgraded from 5.7 to 8.0 on your instance.

If you experience this problem, the following error will be in the mysql error log (`/var/log/mysql/mysql.err`):

Shell

```
[ERROR] [MY-012526] [InnoDB] Upgrade after a crash is not supported. This redo
log was created with MySQL 5.7.40. Please follow the instructions at
http://dev.mysql.com/doc/refman/8.0/en/upgrading.html
```

Avoiding this issue

We strongly recommend you upgrade your GitHub Enterprise Server instance to the latest patch version (3.7.14 or higher, or 3.8.7 or higher) before you upgrade to 3.9 or 3.10. These versions contain a fix for the upgrade issue.

If you cannot upgrade your GitHub Enterprise Server instance, then you can avoid the issue by updating the nomad timeout for MySQL before starting an upgrade to GitHub Enterprise Server 3.9 (from 3.7 or 3.8) or 3.10 (from 3.8 only).

1. Put your instance into maintenance mode:

```shell
ghe-maintenance -s
```

2. Update consul template for nomad:

```shell
sudo sed -i.bak '/kill_signal/i \      kill_timeout = "10m"' /etc/consul-templates/etc/nomad-jobs/mysql/mysql.hcl.ctmpl
```

3. Render consul template for nomad:

```shell
sudo consul-template -once -template /etc/consul-templates/etc/nomad-jobs/mysql/mysql.hcl.ctmpl:/etc/nomad-jobs/mysql/mysql.hcl
```

4. Verify current `kill_timeout` setting:

```shell
nomad job inspect mysql | grep KillTimeout
```

   Expected response:

```shell
"KillTimeout": 5000000000
```

5. Stop MySQL:

```shell
nomad job stop mysql
```

6. Run new MySQL job:

```shell
nomad job run /etc/nomad-jobs/mysql/mysql.hcl
```

7. Verify kill_timeout has been updated:

Shell

```
nomad job inspect mysql | grep KillTimeout
```

Expected response:

Shell

```
"KillTimeout": 600000000000,
```

8. Take instance out of maintenance mode:

Shell

```
ghe-maintenance -u
```

Now that the nomad timeout for MySQL has been updated you can upgrade your GitHub Enterprise Server instance to 3.9.

Mitigating a failed restart of MySQL

If you're affected by this problem, restore your GitHub Enterprise Server instance to the state it was in prior to the upgrade attempt, and then follow the steps from the previous section.

For more information about restoring from a failed upgrade, see "Upgrading GitHub Enterprise Server."