

FROM PLANNING TO PERFORMANCE:

MongoDB Upgrade Best Practices

From Planning to Performance: MongoDB Upgrade Best Practices

In database management, keeping pace with the ever-advancing landscape of technologies and features isn't just a choice; it's essential. This holds especially true for MongoDB users, as it continuously evolves to deliver enhanced performance, heightened security, and unmatched scalability. However, upgrading MongoDB isn't a matter of just clicking a button; it demands meticulous planning, exacting execution, and an intricate understanding of how upgrades work.

Whether you're a seasoned DBA well-versed in MongoDB or a newcomer looking to harness its potential, **this ebook provides the insights, strategies, and best practices to guide you through MongoDB upgrades**, ensuring they go as smoothly as possible and your databases remain optimized and secure.

Why you should upgrade MongoDB

Not too concerned about upgrading your MongoDB instances? You may want to be! Outdated versions may lack critical security updates, optimizations, and new features that could result in data loss, slower response times, and reduced efficiency, creating more work for you in the future. To illustrate, take a look at just a few features that have been included in each new MongoDB version.

In MongoDB 5.x

- Introduction of the capability for live resharding of databases, meaning users can change the sharded key for collections as their workloads and databases grow and evolve without experiencing any database downtime.
- Time-series collections can now be sharded, offering improved scalability and performance for managing temporal data.
- MongoDB 5.3 saw a new feature called clustered collections, where data is stored according to the keys in the associated clustered index rather than in the order in which it was inserted. While they may have slightly slower write performance, they excel when analytical queries require data to be in the specific order defined by the clustered index.

In MongoDB 6.x

- Time-series collections in 6.0 now include secondary and compound indexes on measurements, which improve read performance and new use cases like geo-indexing.
- MongoDB 6.0 enhances change streams significantly, and users can now access the before and after state of changed documents, enabling tasks like updating entire documents downstream and referencing deleted ones. They also now support data definition language (DDL) operations, such as creating or dropping collections and indexes.
- Administrators can now compress and encrypt audit events before saving them to disk, utilizing their own KMIP-compliant key management system.

In MongoDB 7.x

- The ability to modify time-series data provides greater flexibility and control over your time-based information.
- New operators and variables for aggregation pipelines for more complex and sophisticated data transformations.
- Wildcard indexes optimize the execution of queries, resulting in faster and more efficient searches.
- Queryable encryption allows you to query encrypted data while maintaining data confidentiality.

Beyond the ability to take advantage of new features, upgrading MongoDB can have a direct – and positive – impact on your database’s performance, security, and scalability:



Security enhancements

One of the standout benefits is enhanced security. With each MongoDB version release, you gain access to essential security enhancements that address vulnerabilities discovered in previous versions and introduce advanced features for authentication and encryption. In addition, older MongoDB versions that have reached End of Life are no longer supported, making them more vulnerable to security threats.

By staying aware of updates, you can help ensure your data remains well-protected against evolving threats.



Improved performance

Another significant advantage of upgrading MongoDB is improved performance. MongoDB continually fine-tunes its database engine, which translates to a more efficient and responsive database, resulting in faster query execution, reduced latency, and an enhanced user experience. And, as your data requirements expand, you can take advantage of improved scalability features typically included in each version release.



Ensuring regulatory compliance

Upgrading MongoDB is a critical step in ensuring compliance with various legal and regulatory standards. Using outdated or End of Life software can expose your organization to the risk of violating these requirements. While the specifics vary across industries, the financial sector, in particular, demands a careful approach to data management and security. By keeping MongoDB up to date, you not only reduce the likelihood of legal complications but also demonstrate a commitment to upholding rigorous standards of data integrity and security.



Scalability improvements

Upgrading to a more recent MongoDB version allows you to utilize advanced scalability features, ensuring the seamless expansion and performance of your database to meet evolving requirements. This becomes pivotal when your application experiences traffic growth, increased user concurrency, or the diverse requirements of a global customer base. This isn’t just for addressing your immediate needs – it also protects against late-night scrambles to meet future demands that may pop up.



Protection against data loss

The loss of data is a critical concern associated with the usage of older MongoDB versions. As these versions may no longer be supported and have unresolved bugs, they can introduce data corruption or loss scenarios related to database structure and data format compatibility with modern backup and recovery tools. By keeping MongoDB up to date, you can take advantage of efficient and reliable backup processes to safeguard against data loss while enhancing overall database resilience.

When considering an upgrade, it's essential to not only know why you are upgrading but also to have a clear understanding of what to expect during the process. **MongoDB upgrades follow a well-documented and structured approach.** It begins with comprehensive planning, where you assess the specific features and improvements in the new version that align with your database's unique needs. This is followed by thorough testing in a controlled environment, checking compatibility with your existing applications, creating a comprehensive backup of your current database, and being aware of common complications, all crucial steps for a smooth and efficient MongoDB upgrade.



Assessing your MongoDB environment

Before you dive in and begin upgrading MongoDB, it's vital to conduct a comprehensive evaluation of your existing deployment. This initial step is often overlooked but is pivotal in ensuring a smooth and successful upgrade.

Here's what you need to be aware of to minimize the risks — *and maximize the benefits* — of upgrading your MongoDB environment.

First, start by identifying which version of MongoDB you are currently using. Because each version release of MongoDB brings different improvements and features, knowing your current version is essential for determining the gap between your existing setup and the latest version. Knowing this information is vital because when it's time to do the upgrade, we strongly recommend that you follow a specific upgrade path between versions. We will cover this later in greater detail; **be sure to note it during your planning.**

Once you know what version you are running and what your target version is, it's time to check out what your instance runs on, how your organization uses MongoDB, any custom configurations you have in place, and the compatibility of your applications with a new version.

Resources and hardware

Are your current servers equipped with sufficient CPU, memory, and storage capacities to handle the new version efficiently? It's important to note that upgrading MongoDB may necessitate adjustments to your hardware specifications to meet the prerequisites of your target version. And thinking ahead long-term, will the new setup meet your future workloads and scaling needs? The last thing you want to do is only minimally upgrade your hardware or resources, only to find you have to do it again a few months later. Why create more work for yourself?

Understand how you use MongoDB

How does your organization utilize MongoDB? Because of how versatile it is, MongoDB can be used for a wide range of purposes. Is it primarily used for transactional data? Analytical queries? Gaming applications? Different versions of MongoDB may be better suited for specific use cases, so it's essential to evaluate if the new target version aligns with your database's intended purpose.

In addition, check if your applications and drivers are compatible with the MongoDB version you want to upgrade to. Incompatibilities with your application's tech stack could lead to service interruptions and, worse, complete application failures. Compatibility is essential for a smooth upgrade.

Watch for custom configurations and settings

This part is essential if you have customized any MongoDB configurations to meet your specific requirements. Because each new version of MongoDB often includes changes to configuration options and settings, some of your customizations may no longer work. To save yourself from unexpected issues, ensure that your custom settings are compatible with the target MongoDB version — or plan to adjust them accordingly. You can review the MongoDB documentation for your current and new versions to see what will work... and may not.

Prepping for an upgrade: Setting the stage

Upgrading MongoDB is a significant undertaking that demands careful planning and execution. Having a well-structured upgrade plan serves as your roadmap to a successful MongoDB upgrade, covering the entire process, including tasks, timelines, dependencies, and communication about any downtime that may affect your organization's operational requirements.

In this chapter, we'll explore the pivotal steps required to get your MongoDB upgrade off to a successful start.



Safeguard your data

Data is the lifeblood of not only your MongoDB database but your entire organization. Before implementing any changes, executing a thorough backup of your current data is vital. This backup should encompass not only your database's contents but also extend to your application data, customizations, replication configurations, indexes, and security settings. You can utilize MongoDB's built-in backup tools or opt for a trusted third-party solution to take a full backup.



Test, test, and test in a sandbox environment

Testing is an essential phase that should always be addressed. Before you even consider implementing the upgrade in your production environment, it's imperative to take the time to create a separate sandbox or staging environment that closely mimics your production setup. This environment is a controlled and isolated testing ground for the upgrade process.

In this staging environment you have created, you should subject the upgrade process to a thorough and systematic examination. The primary objective here is to scrutinize every upgrade aspect, identifying any issues, conflicts, or unexpected behaviors that may arise during the transition, allowing you to catch potential problems in a safe and controlled setting. This testing phase should not be a one-time affair. It should occur repeatedly, with each cycle an opportunity to fine-tune every upgrade aspect, ensuring it operates without disruption. *Always remember the value of thorough testing.*



Have rollback plans

While careful planning and following best practices is essential for a successful MongoDB upgrade, so is preparing for any challenges. If something does go wrong, it's crucial to have a way to roll back to the previous version to avoid extended downtime and the potential for data loss.

A comprehensive MongoDB rollback plan is typically comprised of the following key components:

- Data and configuration backup: As mentioned earlier, ensure you have an up-to-date backup of your MongoDB data, covering databases, collections, indexes, configurations, and essential files.
- Document the current state: Thoroughly document the existing state of your MongoDB deployment.

- **Communication plan:** Understand who needs to be informed of the decision to execute a rollback and the potential for downtime.
- **Identify the rollback trigger:** Establish clear criteria for a rollback, including specific errors, performance issues, or other outcomes.
- **Rollback procedure:** Create a step-by-step approach for executing a rollback, including instructions for stopping the MongoDB service, restoring data from your backup, and resuming normal operations.
- **Know what to monitor:** During and after the rollback, monitor your MongoDB environment to verify it was successful and watch for any continuing issues.
- **Analyze:** After the rollback, determine what caused the upgrade failure to avoid it in future upgrade attempts.



Consider the use of stable vs. development versions

Stable versions of MongoDB have typically undergone extensive testing and are deemed production-ready, whereas those still in development may not be fully ready for prime time. Your choice between these versions should be guided by your organization's risk tolerance and your upgrade's specific objectives. In this decision-making process, having a clear understanding of your current MongoDB version and the version you're targeting becomes exceptionally valuable.

By understanding your current environment, including any unique configurations and requirements, you can make an informed decision about whether to opt for a stable, production-ready version or consider a version still in development that may offer exciting new features or improvements.



Track your progress

During this preparation phase, it's crucial to implement monitoring mechanisms to keep a close eye on the progress of your upgrade plan so you can regularly assess the status of your plan and ensure it stays aligned with your goals and objectives. Monitoring and reviewing your plan consistently serves as a proactive measure to identify any issues, enabling you to make necessary adjustments and maintain a smooth MongoDB upgrade.



Communicate with key stakeholders

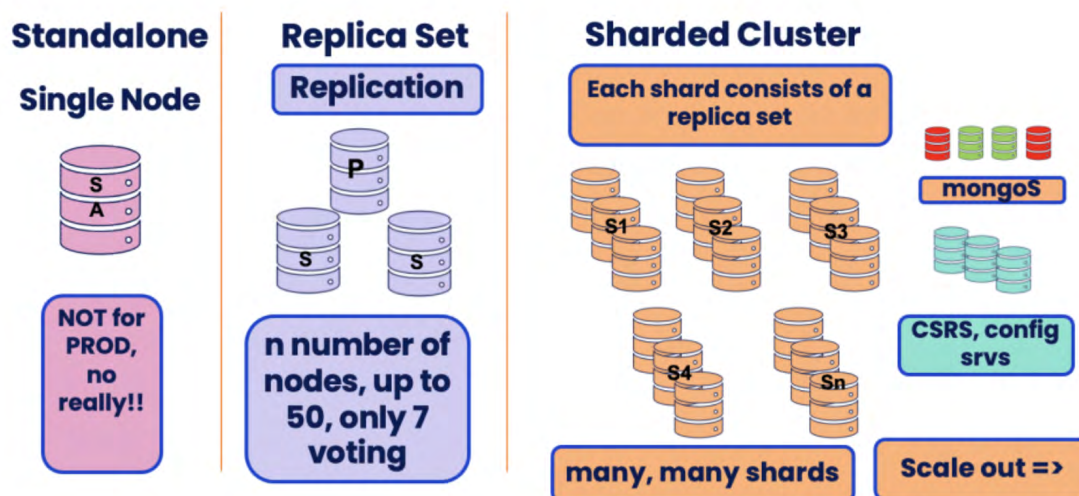
Notify all relevant stakeholders about the upcoming upgrade, including database administrators, developers, and end-users. Clear communication helps manage their expectations, minimizes disruptions, and instills confidence in how the upgrade process will go, so make sure they are fully apprised of the schedule and any potential for downtime.

With these essential preparatory steps in place, you'll be ready for your MongoDB upgrade. The next chapter will look at a general upgrade path to ensure a seamless move.

The upgrade process

While the upgrade process varies, of course, depending on each individual environment, here we will provide a general path for a basic MongoDB upgrade as well as considerations for standalone, replica set, and sharded cluster architecture variations.

Types of MongoDB Systems



Basic overall MongoDB upgrade steps

- Take a backup
- Download new desired binaries
- Keep FCV (Feature Compatibility Value) set to Current/Previous version
- Shut down mongo processes in the correct order; rolling fashion according to system type
- Replace current binaries with new binaries
- Start up mongo processes in the correct order; rolling fashion as required by system type
- Wait for an amount of time to ensure no problems
- Set FCV to the new desired version

Upgrading a standalone system

- Take a backup
- Shut down the mongod process
- Replace the current binary with the new binary
- Keep FCV at the previous version until everything checks out after the upgrade
- Restart the mongodb process using the original port (default 27017)
- Check version, data, logs, etc.
- When all is good, update FCV

Upgrade major version in a replica set

- Before upgrading, take the backup/copy of the MongoDB configuration file on every node, as it will be replaced with the default one.
- Start upgrading with secondaries one by one
- Stop the service on one secondary (if not logged in as root user, make sure the user has sudo access)
- After the service is stopped successfully, remove the binaries
- Enable the repository
- Install specific version packages
- Before restoring, make sure to remove the content of the newly created configuration file
- Start the mongod service
- Check the installed version
- Wait for the secondary to get in sync and enter the "SECONDARY" state.
- Now perform the same steps one through nine, including taking a backup/copy of the config file on other nodes.
- Once both the secondaries are upgraded, log in to the Primary node and stepDown it.
- Once a new primary is elected, repeat the same steps.
- Change the FCV to the current version once everything is running fine.

Upgrade major version in sharded cluster

- Stop the balancer
- Upgrade the config server replica set first. Refer to previous steps from the upgrade major version in the replica set.
- Upgrade the shard replica sets next. Refer to the upgrade major version in the replica set

section. Also, you can upgrade the shards in parallel in the same rolling manner. The steps will be the same for all shards.

- The last mongos will be upgraded in the same rolling manner (If you have one mongos, please inform your application team of that downtime).
- Start the balancer.

Upgrade minor version in a replica set

- Before starting the upgrade, take the backup/copy of the configuration file (mongod.conf) file on every node, as it will be replaced with the default one. Use the default location of the configuration file for the backup/copy. Start upgrading with secondaries one by one.
- Stop the service on one secondary (if not logged in as a root user, make sure the user has sudo access).
- After the service is stopped successfully, remove the binaries
- Check the available minor versions
- Install the specific version on which you want to upgrade
- Before restoring, make sure to remove the content of the newly created configuration file.
- Start the mongod service
- Wait for the secondary to get in sync and enter the "SECONDARY" state.
- Now perform the same steps, including taking a backup/copy config file on other nodes.
- Once all the secondaries are upgraded, login to the Primary node and stepDown it. Once a new primary is elected, repeat the same steps.
- In the minor version, FCV will remain the same as we are changing the C rather than A or B in A.B.C.

Upgrade minor version in sharded cluster

- Stop the balancer
- Upgrade the config server replica set first. Refer to the steps from the upgrade minor version in the replica set.
- Upgrade the shard replica sets next. Refer to the steps from the upgrade minor version in the replica set. Also, you can upgrade the shards parallel in the same rolling manner; the steps will be the same for all shards.
- The last mongos will be upgraded in the same rolling manner (If you have one mongos, then please inform your application team of that particular downtime).
- Start the balancer

Downgrades

If, for whatever reason, you run into any problem, you can perform a downgrade. Downgrades basically go in reverse order – more applicable to sharded clusters.

- Stop balancer
- Downgrade MongoS's
- Downgrade shard nodes
- Downgrade the config servers
- Restart balancer

While MongoDB upgrades are essential for harnessing the latest features, improving security, and enhancing the performance of your database, the specific upgrade process can vary depending on the unique characteristics of your environment. **It's important to note that our recommended upgrade path for MongoDB is 3.6 > 4.0 > 4.2 > 4.4.1+ > 5.0 > 6.0 > 7.0.** Keep in mind that upgrading multi-sharded cluster infrastructure in production can be complex and challenging and can impose serious risk on your business operations.

Here, we have outlined a general upgrade path to serve as a guide; it is not meant as a diagnostic tool.



Post-upgrade testing and validation

Congratulations on successfully upgrading your MongoDB environment! Upgrading MongoDB is a significant operation, and as you step into the post-upgrade phase, it's crucial to validate that your database is operating seamlessly and per your expectations. Let's look at the best practices for testing and validation to ensure everything functions as it should.

Start your post-upgrade validation with comprehensive functional testing. Examine the basic CRUD (Create, Read, Update, Delete) operations and ensure they work as expected. Pay close attention to any application-specific queries or workflows that could be affected by the upgrade.

Now, it's time to assess the performance of your new MongoDB instance. Compare and monitor key performance metrics like query response times, throughput, and resource utilization with your pre-upgrade environment to identify anything requiring attention. This performance testing can help uncover bottlenecks or inefficiencies introduced during the upgrade.

Is everything working as it should? Great! But what about the data itself?

It's time to verify its integrity by ensuring all documents, collections, and indexes are there and there's no data corruption. You can perform these checks using MongoDB's built-in tools or custom scripts.

Security should be your next concern once you have verified everything is working well, and your data is intact. Conduct security checks of user access controls and permissions to ensure they align with your organization's security policies, and review that authentication and authorization mechanisms are working correctly.

So... your upgraded MongoDB is working fine, performance looks great, your data is intact, and your environment is locked down. Now what? It's time to check that your applications and drivers are still compatible with the new version. Test your applications to ensure they can connect to the database and function as expected without encountering compatibility issues.

Now it's time for the ever-important backup and rollback testing. Verify that you can successfully restore data from your backup to ensure that your data recovery procedures are effective and reliable if you ever need to roll back to the previous version.

Lastly, but certainly not the least important, is making any necessary updates to your documentation to show any changes introduced by the MongoDB upgrade. Ensure that all team members are aware of new features, configurations, or best practices associated with the upgraded version.

To summarize, post-upgrade testing and validation are integral parts of the MongoDB upgrade process that ensure your database is stable, performs optimally, and aligns with your organization's needs. By following these best practices, along with regular monitoring, you can maintain the reliability of your MongoDB environment while taking full advantage of all the benefits of the upgrade.

Next, we will review some of the most common challenges you may face when upgrading MongoDB.

Navigating common MongoDB upgrade challenges

Upgrading your MongoDB environment represents a significant step towards leveraging the latest features, enhancing performance, and upholding the security of your database. Yet, this undertaking frequently presents its own set of challenges. In this chapter, we will delve into the typical issues users encounter when upgrading MongoDB, underscoring the importance of a comprehensive and systematic approach.



Compatibility issues

The rapid evolution of MongoDB introduces new features and modifications with each release. While this signifies progress, it also introduces a potential challenge: compatibility issues with existing applications and drivers. Users often discover their current application stack is incompatible with the target MongoDB version.

These compatibility challenges often stem from the inherent differences between MongoDB versions, including changes in query syntax, data structures, or authentication mechanisms that can affect the stability of the database environment and result in extended downtime, loss of revenue, and a strain on customer relationships. To mitigate these challenges, it's essential to engage in thorough testing and validation processes, seeking to identify and resolve compatibility issues before they reach the production environment.



Data migration problems

One of the most critical aspects of a MongoDB upgrade is the seamless transition of your data to a new version. These issues can take on various forms, from concerns about data integrity to problems with data transformation to match changes in data structures. They can lead to severe problems such as data loss, incorrect query results, application failures, and potential data recovery challenges.

To minimize the chances of experiencing data migration problems, it's advisable to break down your migration into easily-managed parts, using trusted tools for data transfer, and verifying data integrity at each step, along with thorough testing and validation of the migration process *before* applying it to a production environment.



Extended and/or unforeseen downtime

Experiencing prolonged downtime during a MongoDB upgrade can have serious and adverse effects on business operations and customer satisfaction.

Primarily, it can lead to a significant decline in income, as businesses heavily rely on their MongoDB databases for transaction processing and revenue generation. On top of that, downtime can also cost your organization money due to the associated expenses and efforts to restore normal operations.

However, downtime doesn't just affect a company's finances; it can also bring business processes to a halt by slowing productivity and disrupting services. Employees may find themselves without access to essential data and tools necessary to do their jobs, while customers may face frustration attempting to access your system.

Lastly, businesses operating under strict regulatory requirements concerning data availability and integrity may be out of compliance due to extended downtime, which can lead to legal and financial consequences.



Hardware constraints

Insufficient hardware resources can pose a significant obstacle during a MongoDB upgrade, potentially causing performance problems that hinder your database's functionality. Before you start the upgrade, it's vital to assess your hardware requirements thoroughly as part of the preparation phase, ensuring your infrastructure will meet the demands of the new version.

Scaling your resources as needed — before the upgrade — not only ensures a successful upgrade but also maintains the optimal performance of your MongoDB environment.



Complexity of the upgrade process

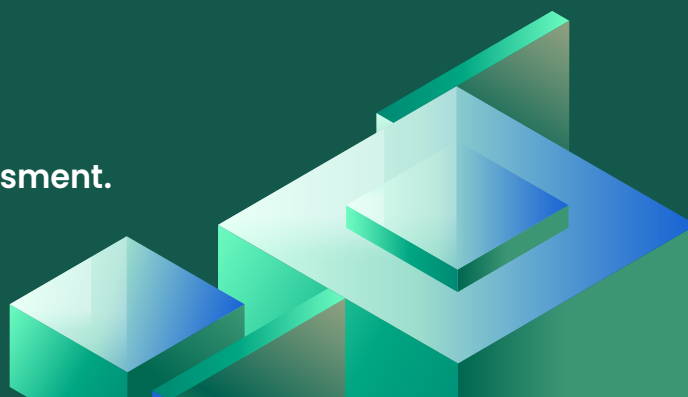
Upgrading MongoDB involves multiple intricate steps, ranging from selecting the correct version to meticulous planning, thorough testing, and, finally, the actual execution of the upgrade itself. This complexity can pose a considerable challenge, particularly for users who may not possess extensive MongoDB expertise. Mistakes made during any upgrade phase can have far-reaching consequences, many of which can happen easily, without notice.

While it's conceivable to embark on a MongoDB upgrade on your own, these challenges underscore the value of having a partner by your side. Percona's wealth of experience and expertise in database management understand the complexities of MongoDB upgrades and mitigating these common issues.

Seeking help with your MongoDB upgrade?

Contact us for an obligation-free assessment.

GET IN TOUCH



Bonus: Optimizing your upgraded MongoDB

Congratulations on a successful MongoDB upgrade! If everything is working as it should, it's now time to focus on optimizing your database to make it even better. This chapter will look at various strategies and best practices to keep your upgraded MongoDB running smoothly.

Your first step should be to **review and fine-tune your MongoDB indexes**. This involves removing unnecessary indexes, optimizing the use of compound indexes, and even crafting custom indexes that are tailored to your most frequently executed queries. Consistently monitoring index usage and performance is essential to maintain peak database performance.

Next, and because MongoDB offers a strong query planner and profiler to identify slow queries, you can **identify areas where queries may be underperforming** and make necessary adjustments to enhance query execution times.

If you operate a MongoDB environment that will experience rapidly growing datasets, you should **implement sharding**. Sharding distributes your data across multiple servers, or shards, effectively transforming your MongoDB deployment into a horizontally scalable system that enhances both read and write scalability along with providing for improved fault tolerance. Careful consideration of your data distribution strategy, sharding key selection, and monitoring of shard health are essential aspects of successfully leveraging sharding in your MongoDB environment.

Regular assessments and **scaling of your MongoDB resources** are essential practices for maintaining a high-performing and efficient database environment. This proactive approach not only safeguards against performance bottlenecks but also ensures cost-effectiveness and future-proofing, allowing your MongoDB deployment to evolve along with your organization's changing needs.

In order to know what's going on with your new MongoDB instances, **implement a monitoring strategy** to watch the health and performance of your deployment closely. Use a tool like [Percona Monitoring and Management](#) to set up alerts for critical events so you can proactively address any issues you see pop up.

On top of your ongoing monitoring efforts, you need to **establish a regular maintenance** schedule for your MongoDB deployment. This includes staying up to date with the latest stable MongoDB version(s), applying security updates and patches, optimizing storage to free up disk space, testing your backup and disaster recovery procedures, and keeping your documentation current so you can share optimization strategies and best practices with your fellow MongoDB team members.

Lastly, be sure to **engage with the MongoDB community**. Join user groups and [forums](#), participate in online discussions, and attend meetups to share your experiences and learn new tips and tricks.

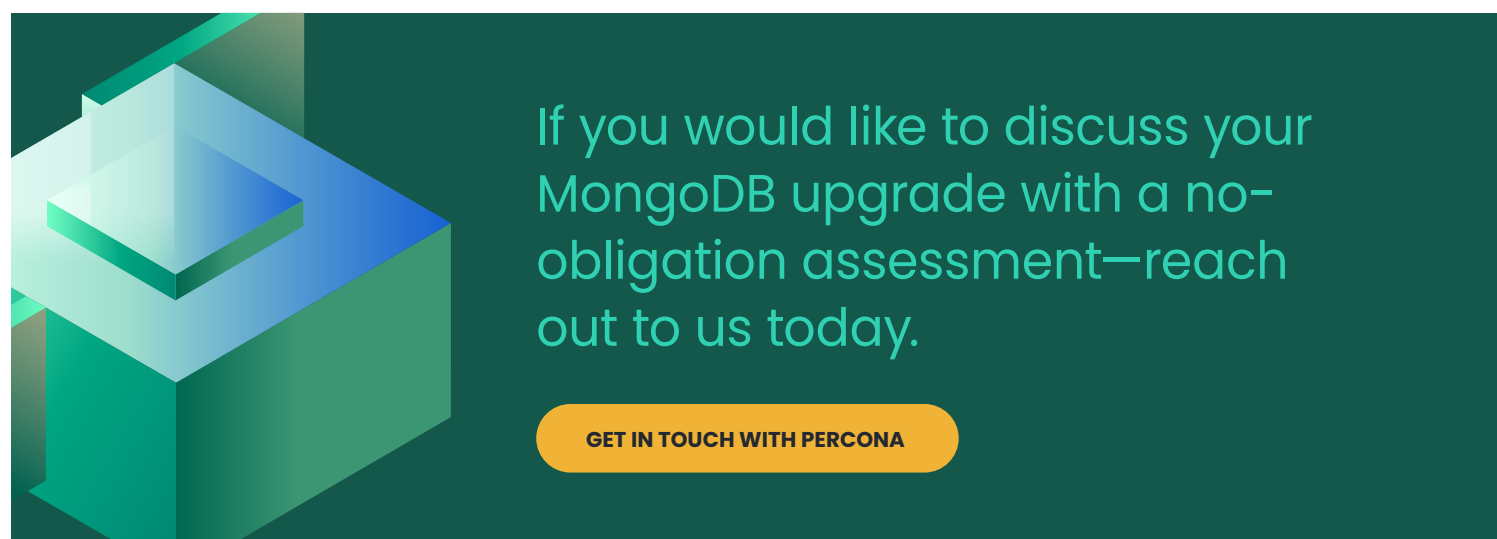
By putting these optimization strategies into action and embracing a proactive approach to database management, you can guarantee that your enhanced MongoDB deployment consistently delivers exceptional performance, scalability, and reliability.

Conclusion

Navigating a MongoDB upgrade can be complex and challenging, but it's a crucial step to optimizing a database's performance, security, and scalability. As you've learned in this ebook, while MongoDB upgrades offer many benefits, upgrades should not be taken lightly; careful planning, testing, and execution are essential, and even for the most seasoned MongoDB administrator, issues arise, and mistakes happen.

But, by following these best practices and maintaining a proactive approach to MongoDB management, you can help to ensure the long-term success of your MongoDB deployment. And if you do need help along the way, Percona is here.

Our expertise in all things MongoDB can guide you through the entire upgrade process, from assessment to post-upgrade optimization. Knowing that you have a trusted partner to assist you in overcoming any challenges that may arise will make your MongoDB upgrade something you look forward to rather than something you keep putting off.





Contact us

For more information about using your database infrastructure to support revenue goals, contact us at +1-888-316-9775 (USA), +44 203 608 6727 (Europe), or via email at sales@percona.com.

