Applying the AWS Well-Architected Framework for Amazon Timestream for InfluxDB

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Applying the AWS Well-Architected Framework for Amazon Timestream for InfluxDB

Table of Contents

Applying the AWS Well-Architected Framework for Amazon Timestream for InfluxDB

Balwanth Bobilli, Amazon Web Services

You can build time-series-based solutions on Amazon Web Services (AWS) by using Amazon Timestream. Amazon Timestream offers fully managed, purpose-built time-series database engines for workloads, from low-latency queries to large-scale data ingestion. With Amazon Timestream for InfluxDB, you can run open source InfluxDB databases on AWS for time-series applications, such as real-time alerting and monitoring infrastructure reliability, with millisecond response times. Timestream for InfluxDB provides up to 99.9 percent availability.

This guide provides prescriptive guidance for applying the AWS Well-Architected Framework principles when you plan your Timestream for InfluxDB deployment. The AWS Well-Architected Framework helps you build secure, high-performing, resilient, and efficient infrastructures for a variety of applications and workloads. It also provides a consistent approach for you to evaluate architectures and implement scalable designs.

The AWS Well-Architected Framework is built around the following six pillars:

- Operational excellence
- Security
- Reliability
- Performance efficiency
- Cost optimization
- Sustainability

This guide provides information from the AWS Well-Architected Framework design pillars. Consider using these best practices when you deploy Amazon Timestream for InfluxDB on AWS.

> **Note**
>
> Some AWS services aren't available in all AWS Regions. For Region availability, see the Service endpoints and quotas page in the AWS documentation, and choose the link for the service.

Intended audience

This guide is intended for data engineers, solutions architects, and data analysts who design and implement solutions for time-series data on AWS.

Objectives

This guide can help you and your organization do the following:

- Choose from the supported deployment options, perform optimized writes, and implement fine-grained access.
- Follow the AWS Well-Architected design patterns that help improve resiliency and security.
- Design your queries for optimal performance.
- Learn how to be operationally efficient when managing your Timestream for InfluxDB instance in production.

Operational excellence pillar

The operational excellence pillar of the AWS Well-Architected Framework focuses on running and monitoring systems, and continually improving processes and procedures to deliver business value. The operational excellence pillar includes the ability to support development and run workloads effectively, and to gain insight into their operation.

You can reduce operational complexity through self-healing workloads, which detect and remediate most issues without human intervention. To work toward this goal, follow the best practices described in this section. Use Amazon CloudWatch metrics for Amazon Timestream for InfluxDB, the InfluxDB native metrics endpoint, APIs, and mechanisms to respond when your workload deviates from expected behavior.

This discussion of the operational excellence pillar focuses on the following key areas:

- Infrastructure as code (IaC)
- Change management
- Resiliency strategies
- Incident management
- Logging and monitoring for auditing purposes

Automate deployment by using an IaC approach

Best practices for automating deployment on Timestream for InfluxDB by using IaC include the following:

- Apply IaC to deploy Timestream for InfluxDB whenever possible. For consistent environment configuration, use an AWS CloudFormation template, AWS Cloud Development Kit (AWS CDK), or HashiCorp Terraform to create all the required resources for your instance.
- Automate Timestream for InfluxDB operational procedures, such as resizing instances.
- Use tags to add metadata to your Timestream for InfluxDB resources, and track usage based on tags. For more information, see Tagging Amazon Timestream for InfluxDB.

Make frequent, small, reversible changes

The following recommendations focus on small, reversible changes to minimize complexity and reduce the likelihood of workload disruption:

- Store IaC templates and scripts in a source-control service, such as GitHub or GitLab. Do not store AWS credentials in source control.
- Require IaC deployments to use a continuous integration and continuous delivery (CI/CD) service, such as AWS CodeDeploy or AWS CodeBuild. These services compile, test, and deploy code in a non-production environment that contains an ephemeral InfluxDB instance before affecting your production InfluxDB instance.
- Test infrastructure and application queries in a lower environment before you deploy them to production. This minimizes the likelihood of a disruption and helps ensure that they perform well with your workload and scale.

Anticipate failure

A self-healing infrastructure exemplifies operational excellence by anticipating failure and attempting to resolve any issues without intervention. The following recommendations help you achieve that maturity with Timestream for InfluxDB:

- Use metrics to monitor your memory, CPU, and storage usage. You can set up CloudWatch to notify you when usage patterns change or when you approach the capacity of your deployment. This way, you can maintain system performance and availability.
- Scale up your DB instance when you are approaching the resource limit. You should have some buffer in storage and memory to accommodate unforeseen increases in demand from your applications.
- If your database workload requires more I/O than you have provisioned, recovery after a failover or database failure will be slow. To increase the I/O capacity of a DB instance, migrate to a different DB instance that has higher I/O capacity.
- If your client application is caching the DNS data of your DB instances, set a time-to-live (TTL) value of less than 30 seconds. The underlying IP address of a DB instance can change after a failover. Caching the DNS data for an extended time can lead to connection failures. Your application might try to connect to an IP address that's no longer in service.

- If your application requires surviving a complete AWS Region outage, consider setting up replication or write to a different Region as part of your disaster recovery (DR) plans. Understand the limitations while setting up replication. For more information about replication, see the InfluxDB documentation.

Learn from all operational failures

A self-healing infrastructure is a long-term effort that you develop in iterations when rare problems occur or responses are not as effective as you want. To focus on achieving a self-healing infrastructure, adopt the following practices:

- Drive improvement by learning from all failures.
- Share what is learned across teams and the organization. If multiple teams within an organization use Timestream for InfluxDB, create a common chatroom or user group to share lessons learned and best practices.

Use logging capabilities to monitor for unauthorized or anomalous activity

To observe anomalous performance and activity patterns, consider the following practices:

- Enable log delivery to store InfluxDB logs in Amazon Simple Storage Service (Amazon S3). InfluxDB logs record information that can help to check the following:
  - Data plane API events
  - Response times
  - Compaction details
  - Any critical errors or warnings encountered by the system

  Review the logs for unauthorized access or anomalies. Overall, logging provides diagnostic information for troubleshooting.
- Timestream for InfluxDB supports logging control plane actions by using AWS CloudTrail. For more information, see Logging Timestream for InfluxDB API calls with AWS CloudTrail.
- You can monitor `CPUUtilization`, `MemoryUtilization`, and `DiskUtilization` metrics from **Timestream/InfluxDB >** <**Namespace**> in CloudWatch.

For more information, see the Timestream for InfluxDB documentation.

Security pillar

The security pillar helps you understand how to apply the shared responsibility model when using Timestream for InfluxDB.

The security pillar includes the following key focus areas:

- Data security
- Network security
- Authentication and authorization

The following sections show you how to configure Timestream for InfluxDB to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Timestream for InfluxDB resources.

Implement data security

Data leakage and breaches put your customers at risk and can cause substantial negative impact on your company. The AWS shared responsibility model applies to data protection in Timestream for InfluxDB. The following practices help protect your customer data from inadvertent and malicious exposure:

- Don't include confidential information in instance names, tags, parameter groups, AWS Identity and Access Management (IAM) roles, and other metadata. That data might appear in billing or diagnostic logs.
- Use encryption to increase data protection of your applications that are deployed in the cloud. Encrypted instances provide an additional layer of data protection by helping to secure your data from unauthorized access to the underlying storage. Timestream for InfluxDB encryption is enabled by default.
- Use parameterization for APIs wherever possible.

For more information about data protection and encryption, see the Timestream for InfluxDB documentation.

Secure your networks

You can create a Timestream for InfluxDB instance only in a virtual private cloud (VPC) on AWS. To further secure your instance, do the following:

- Restrict public access to the instance's endpoint by doing one of the following:
  - Choose **Not publicly accessible** on the **Create InfluxDB database** page (selected by default).
  - Set the PubliclyAccessible property to `false` in AWS CloudFormation (`AWS::Timestream::InfluxDBInstance`).

  When you choose **Not publicly accessible** or set `PubliclyAccessible` to `false`, the instance's endpoints are accessible only within the VPC. The endpoints are usually accessed from an Amazon Elastic Compute Cloud (Amazon EC2) instance running in the same VPC as the Timestream for InfluxDB instance.
- Use security groups to further secure your network access to Timestream for InfluxDB within the VPC.
- Use SSL/TLS to communicate with AWS resources. Timestream for InfluxDB requires TLS 1.2, and we recommend TLS 1.3.
- Securely connect to your instance by following the instructions in Creating and connecting to a Timestream for InfluxDB instance.
- Establish a private connection between your VPC and Amazon Timestream for InfluxDB control plane API endpoints by creating an interface VPC endpoint.

For more information about security, see the Timestream for InfluxDB documentation.

Implement authentication and authorization

Use IAM credentials to control management actions on Timestream for InfluxDB instances. When you connect to Timestream for InfluxDB by using IAM credentials, your IAM role must have IAM policies that grant the permissions required to perform Timestream for InfluxDB management operations. Ensure that you follow the principle of least privilege, granting only the permissions required to complete a task. For more information, see Identity and Access Management for Amazon Timestream for InfluxDB.

You can use Amazon Timestream for InfluxDB actions in the IAM policy to control who can manage your Timestream for InfluxDB instance.

To provide fine-grained access control for your data stored in your Amazon Timestream for InfluxDB instance, give users InfluxDB API tokens.

For interacting with other AWS services, Amazon Timestream for InfluxDB uses IAM service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Timestream for InfluxDB. Service-linked roles are predefined by Timestream for InfluxDB, and they include all the permissions that the service requires to call other AWS services on your behalf. For more information, see Using Service-Linked Roles for Amazon Timestream for InfluxDB.

Reliability pillar

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to. This includes the ability to operate and test the workload through its complete lifecycle.

Configuring a reliable workload starts with upfront design decisions for both software and infrastructure. Your architecture choices will impact your workload behavior across all of the Well-Architected pillars. To achieve reliability, you must follow specific patterns.

The reliability pillar focuses on the following key areas:

- Workload architecture, including service quotas and deployment patterns
- Managing and scaling InfluxDB instances

Workload architecture, including service quotas and deployment patterns

Each AWS account has quotas for resources offered in each AWS Region. For example, each Region has a quota for Timestream for InfluxDB instances, regardless of instance size. After you reach the maximum number of instances in a Region, additional calls to create instances fail with an exception. A Timestream for InfluxDB instance storage volume can grow to a maximum size of 16 tebibytes (TiBs) in all supported AWS Regions.

Deployment patterns

For high availability and failover support for Timestream for InfluxDB instances, you can use Multi-AZ deployments with a single standby DB instance. This type of deployment is called a Multi-AZ DB instance deployment. Amazon Timestream for InfluxDB uses the Amazon failover technology. In a Multi-AZ DB instance deployment, Amazon Timestream automatically provisions and maintains a synchronous standby replica in a different Availability Zone. To provide data redundancy, the primary DB instance is synchronously replicated across Availability Zones to the standby replica.

Running a DB instance with high availability can provide availability during DB instance failure or Availability Zone disruption. If an unplanned outage of your DB instance results from an infrastructure defect, Amazon Timestream for InfluxDB automatically switches to the standby

replica. The time that it takes for the failover to complete depends on the database activity and other conditions at the time that the primary DB instance became unavailable.

Failover times are typically 60–120 seconds. However, large transactions with high-cardinality data or a lengthy recovery process with pre-warmup requirements can increase failover time. After the failover is complete, additional time might be required before the Timestream console reflects the new Availability Zone.

If your application must remain available during a complete AWS Region outage, consider setting up replication or writing to a different Region as part of your disaster recovery (DR) plans. However, before you set up replication, be sure that you understand the limitations. For more information, see the InfluxDB documentation.

Amazon Timestream for InfluxDB periodically takes internal backups and retains them for 24 hours to support availability and durability. Snapshots are taken during deletes and retained for 30 days to support restores. To access or use these, create a case at AWS Support.

Manage and scale Timestream for InfluxDB

Timestream for InfluxDB supports instance classes that are ideal for running memory-intensive workloads in open source InfluxDB databases. The different db.influx instance classes have limits on vCPUs, memory, storage, and network bandwidth. To choose the instance class that fits your application's write and query latency requirements, observe the Amazon CloudWatch `CPUUtilization`, `MemoryUtilization`, and `DiskUtilization` metrics during testing. You can scale your instances up and down based on your workload requirements. Timestream for InfluxDB provides multiple storage tiers that are preconfigured with optimal IOPS and throughput required for different types of workloads. Choose what works best for your workload based on your requirements.

If your scaling needs change at predictable times, you can use an AWS Lambda function or a custom scheduler and run an API or SDK to scale up and down with some buffer time.

You manage your InfluxDB configuration in Timestream for InfluxDB by using parameters in a parameter group. Parameter groups act as a *container* for InfluxDB configuration options that are applied to one or more DB instances. When modifying parameters in parameter groups, understand the difference between static and dynamic parameters, and how and when they are applied. To see the current applied configuration, use the GetDbParameterGroup API action.

Performance efficiency pillar

The performance efficiency pillar of the AWS Well-Architected Framework focuses on how to optimize performance while ingesting or querying data. Performance optimization is an incremental and continual process of the following:

- Confirming business requirements
- Measuring the workload performance
- Identifying underperforming components
- Tuning the components to meet your business needs

The performance efficiency pillar provides guidelines that can help you choose a high-performing data model. The performance efficiency pillar includes query and write optimization best practices.

The performance efficiency pillar focuses on the following key areas:

- Influx data modeling and query optimization
- Write optimization

Influx data modeling and query optimization

Designing an effective schema is crucial for optimizing the performance and querying capabilities of time-series data in InfluxDB. Start by choosing the right tags and fields. InfluxDB indexes tags, so the query engine doesn't need to scan every record in a measurement to locate a tag value. This means that querying tags is more efficient than querying fields. To compact and store data, the storage engine groups field values by series key, and then it orders those field values by time. A series key is defined by measurement, tag key and value, and field key. For more information about data design, see the InfluxDB documentation.

The storage engine uses a Time-Structured Merge Tree (TSM) data format. For more information about the TSM data format, see the InfluxDB documentation..

Imagine that you're collecting data (`timestamp`, `host_id`, `region`, `cpu`, `memory`, `network_in_bytes`, `network_out_bytes`, `disk_io`) as part of a DevOps use case. Tags, including the record timestamp, provide context to help identify the who, what, when, and where of a record. Tags are used to organize and categorize data, and to filter data as part of a query.

The `host_id` and `region` tags are ideal tags for organizing and categorizing the DevOps use case. These columns help to filter the data for particular host or to run analysis based on the region column.

Measures provide the basis for performing mathematical calculations (such as computing totals, averages, and differences in rate of change) and quantitative analysis on your data. Therefore, `cpu`, `memory`, `network_in_bytes`, `network_out_bytes`, and `disk_io` capture important metrics related to the DevOps that are changing over time. You can use these metrics to perform various analyses, such as calculating the CPU and memory across different hosts. You can use these metric values to make data-driven decisions that help with avoiding production outages and performing infrastructure planning.

Cardinality is the combination of unique tag values. Aim to keep the cardinality as low as possible, If your application requires a unique identifier for each data point, use field values instead of tag values. This will result in significantly better query latency. Good schema design can prevent high series cardinality, resulting in better performing queries. If you notice data reads and writes slowing down or you want to learn how cardinality affects performance, see the Timestream for InfluxDB documentation.

If your application emits JSON objects, convert them to individual columns (tags or fields), and load the columns into InfluxDB. InfluxDB is designed for time-series data, so organizing your data with individual columns is a best practice for taking full advantage of the service's capabilities.

A single InfluxDB v2.7 OSS instance supports approximately 20 InfluxDB buckets actively being written to or queried across all organizations. More than 20 buckets can adversely affect performance. There are limits on some InfluxDB configuration options, and there are some options that you can configure based on your use case. Validate the configuration based on the application workload during the testing phase. Data retentions are configured at the bucket level, so data with different data-retention requirements should be stored in different buckets. For more information about configuration options, see the Timestream for InfluxDB documentation.

Store data in tag values or field values, not in tag keys, field keys, or measurements. If you design your schema to store data in tag and field values, your queries will be easier to write and more efficient. For more best practices on data modeling, see Design for performance.

Use InfluxDB tasks to pre-aggregate data, load the data into different measurements or buckets, and generate data for dashboards and visualizations from them.

InfluxDB OSS exposes a `/metrics` endpoint that returns performance, resource, and usage metrics formatted in the Prometheus plain-text exposition format. Use InfluxDB templates to set up monitoring and alerting to proactively detect issues, such as high query latency, write throughput degradation, or resource usage spikes.

Timestream for InfluxDB provides Influx IO Included storage. Selecting the appropriate IOPS size can significantly speed up query execution. This is especially helpful for queries that need to scan large amounts of data or handle a high range of requests. In some situations, a combination of scaling up the instance and enhancing the IOPS might be necessary to achieve the performance improvements that you want.

We recommend matching the dev and prod environments (instance class, storage type, configurations). Test changes in the lower environment for every release before moving to production. On Influx IO Included storage volumes, Timestream for InfluxDB provides three storage tiers that are preconfigured with optimal IOPS (3,000, 12,000, 16,000) and throughput required for different types of workloads. Most use cases require less than 3,000 IOPS. Choose 12,000 or 16,000 only if performance testing indicates a need for high IOPS. For more information, see the Setting up section in the Timestream for InfluxDB documentation.

Optimize writes

To optimize writes to InfluxDB, we recommend writing data in batches of 5,000 lines of line protocol per request to minimize network overhead. For better performance, sort tags by key in lexicographic order before writing data points. Using the coarsest time precision possible for timestamps, instead of nanoseconds, can also improve performance. Enabling gzip compression is another way to speed up writes and reduce network bandwidth. In the `influxdb_v2` output plugin configuration in your `telegraf.conf` file, set the `content_encoding` option to `gzip`. Implementing these optimizations can significantly improve the performance and efficiency of writing data to InfluxDB. For more InfluxDB write best practices, see Optimize writes to InfluxDB.

InfluxDB' s write performance is often closely tied to the available IOPS. When writing data, InfluxDB needs to perform a significant number of I/O operations to store the data. When you increase the IOPS, InfluxDB can process more writes per second.

Cost optimization pillar

The cost optimization pillar of the AWS Well-Architected Framework focuses on avoiding unnecessary costs and building architectures in a cost-optimized way. The following recommendations can help you meet the cost optimization design principles and architectural best practices for Amazon Timestream for InfluxDB.

The cost optimization pillar focuses on the following key areas:

- Understanding your use case's requirements and costs
- Selecting resources with attention to cost
- Scaling to meet business needs without overspending
- Right-sizing data storage and transfer

Understanding your use case's requirements and costs

We recommend not using Timestream for InfluxDB in the following use cases:

- If your data model has relational data, Timestream for InfluxDB is not the right solution.
- If you cannot use time filters in your queries, Influx will scan all series, which is inefficient.

Selecting resources with attention to cost

InfluxDB instance costs are based on an hourly rate for the hours that your instance runs. Instances make up, on average, 85 percent of the overall cost of running a database on AWS, so right-sizing can have significant cost implications. The best way to right-size instances is to test application performance:

- Are the `CPUUtilization` and `MemoryUtilization` constantly high or low?
- What is the balance between price and performance?

Instance costs scale linearly. The hourly cost of the `db.influx.2xlarge` instance is twice that of the `db.influx.xlarge` instance, although it also has twice the resource allocation. The `db.influx.16xlarge` instance is 16 times the hourly cost of the `db.influx.xlarge` instance.

Estimate your workload's number of writes and reads for a specific time frame (second, minute, hour, or day). Timestream for InfluxDB instances support 50,000 to more than 500,000 writes per second and 10–100 queries per second (QPS) based on the instance type. For example, `db.influx.2xlarge` typically supports up to 150,000 writes per second and approximately 25 QPS. With an efficient data model and efficient querying, it can exceed that performance. If your requirements vary by the time of day, week, or month, you can schedule scaling up and down by doing the following:

- Create and schedule an AWS Lambda function.
- Use a custom scheduler and run an API or an SDK to scale up and down with some buffer time.

Scaling to meet business needs without overspending

For entry-level experimentation with Timestream for InfluxDB, you can use `db.influx.medium` and `db.influx.large`. These instances are large enough for you to get experience with Timestream for InfluxDB before you invest in larger instances.

The `db.influx.medium` and `db.influx.large` instances are good for low-cost development environments. However, they have a smaller RAM (8 GiB and 16 GiB), fewer vCPUs (1 vCPU and 2 vCPUs), and network performance up to only 10 GB. Not all workloads are suitable for these instance classes. Monitor `CPUUtilization` and `MemoryUtilization`, and scale up or down as needed. There is often a consistent ratio between memory and vCPU. The db.influx instance class has a memory-to-vCPU ratio similar to the Amazon EC2 r7g instance class. We strongly recommend running end-to-end performance or load testing before going to production.

Efficient data modeling, batch writing, and optimized queries require less memory and compute usage. When less resources are required, you can potentially use smaller instances.

Right-sizing data storage and transfer

For storing data, use the following best practices:

- Store only time-series data in Timestream for InfluxDB.
- Set appropriate retention on the InfluxDB bucket so that data older than the retention is deleted, and shards are periodically compacted automatically. For more information, see the InfluxDB documentation.
- Optimize disk usage for future writes.

- Delete any InfluxDB buckets that are not required for your workloads. InfluxDB supports deletes. You can perform scheduled cleanups if that fits your use case.

For data transfer, we recommend deploying your application in same AWS Region as your Timestream for InfluxDB database instance to avoid cross-Region network overhead. There might also be data transfer charges. For more information about data transfer, see the pricing page.

Sustainability pillar

The sustainability pillar focuses on minimizing the environmental impacts of running cloud workloads. The sustainability pillar contains the following key focus areas:

- Understanding your impact
- Sustainability goals
- Maximizing use to minimize resources
- Anticipating and adopting new, more efficient hardware and software offerings
- Using managed services
- Reducing downstream impact

This guide focuses on understanding your impact. For more information about the other sustainability design principles, see the AWS Well-Architected Framework.

Your choices and requirements have an impact on the environment. To increase the sustainability of your workload, do the following:

- Choose AWS Regions that have lower carbon intensity.
- Size your resources to reflect actual workload needs instead of maximizing uptime and durability.
- Optimize your data model and maximize compute resource use.

The next sections discuss practices that you can adopt to reduce environmental impact in your workload design and ongoing operations.

AWS Region selection

Some AWS Regions are near Amazon renewable energy projects or located where the grid's published carbon intensity is lower than other grids. Evaluate Regions based on your sustainability goals and your workload requirements. Then cross-reference your list of viable Regions with the Regions where Timestream for InfluxDB is available.

Base resource consumption on user-behavior patterns

Right-sizing your consumption to match the traffic and behavior of your users helps AWS minimize the impact of services on the environment. When designing your solution, consider the following best practices:

- Monitor Amazon CloudWatch metrics such as `CPUUtilization` and `MemoryUtilization` to determine when your demand is highest and lowest. Ensure that your instance resources are right-sized during those times.
- Consider aligning your service-level agreements with sustainability goals in addition to business continuity goals. Easing requirements such as multi-Region disaster recovery, high availability, or long-term backup retention can reduce the amount of resources required to meet those goals.. Non-production environments and non-mission critical workloads provide opportunities to reduce requirements.

Optimize software development and architecture patterns

To prevent waste, optimize your data model and queries. Share compute resources so that you use all the resources that are available in the Timestream for InfluxDB instance. We recommend implementing the following best practices:

- Encourage developer teams to share the Timestream for InfluxDB stack for better usage of resources wherever possible.
- Implement patterns that maximize the use of resources and minimize idle time. Pattern examples include using parallel threads to load data and batching records together into a larger transaction.
- Optimize your queries and InfluxDB data model to minimize the resources required to compute the results.
- Use InfluxDB tasks to pre-aggregate the data and reduce the scanning of the same raw data by different users for visualizing or dashboarding.
- Keep your Timestream for InfluxDB environments up to date. The newest versions of Timestream for InfluxDB support the latest EC2 instances, such as Graviton, that are more efficient. The newest DB versions also include query optimization improvements and bug fixes that reduce the amount of resources needed to calculate your queries.

Resources

References

- AWS Well-Architected
- AWS Well-Architected Framework documentation
- Amazon Timestream for InfluxDB documentation
- InfluxDB OSS v2 documentation

Blog posts

- Use the AWS InfluxDB migration script to migrate your InfluxDB OSS 2.x data to Amazon Timestream for InfluxDB
- Run and manage open source InfluxDB databases with Amazon Timestream

Document history

The following table describes significant changes to this guide.

Change	Description	Date
Initial publication	—	January 28, 2025