### **Risks Categorized According to the Document**

#### **Resource Management**

* **Risk: Inadequate Cluster Sizing for Production Workloads (Low Severity)**
  + **Architecture Component:** MongoDB Instances (Data Plane).
  + **Details:** The document specifies M10 clusters for non-production and M30 for production (see "Tenancy" section). If Application Teams provision smaller clusters (e.g., M5) for production workloads due to cost-saving, it could lead to performance issues or downtime, as smaller clusters may not handle production-scale traffic.
  + **Cause:** Lack of automated validation during provisioning to enforce minimum cluster sizes.
* **Risk: Lack of Resource Inventory in ME Privacy Inventory (Medium Severity)**
  + **Architecture Component:** MongoDB Instances (Data Plane).
  + **Details:** The document requires all data elements to be registered in a firm-approved inventory (see "Inventory/Data Classification" section). If MongoDB instances or their data classifications aren’t tracked in the ME privacy inventory, it could lead to untracked sensitive data, increasing the risk of breaches or non-compliance with GS policies.
  + **Cause:** Inconsistent inventory management processes or lack of integration with ME tools.

#### **IAC/SDLC**

* **Risk: Misconfiguration of SkyFoundry/Terraform Onboarding (Medium Severity)**
  + **Architecture Component:** MongoDB Atlas Control Plane.
  + **Details:** The document specifies that SkyFoundry/Terraform should not be used for provisioning (see "IaC/SDLC" section). However, if teams accidentally use Terraform or SkyFoundry for onboarding due to legacy practices, it could lead to unsupported configurations, violating GS provisioning standards and introducing security gaps.
  + **Cause:** Lack of strict governance or automation to enforce CFT-only provisioning.
* **Risk: Inconsistent IAM Role Creation for Multi-Region Deployments (Medium Severity)**
  + **Architecture Component:** MongoDB Atlas Control Plane (IAM roles for multi-region).
  + **Details:** The document requires IAM roles to be created for multi-region deployments (see "IaC/SDLC" section). If these roles are inconsistently applied or misconfigured across regions, it could disrupt connectivity or force fallback to less secure methods, like public endpoints.
  + **Cause:** Manual errors during role creation or lack of automation for multi-region setups.

#### **Network Connectivity to MongoDB - Data Plane**

* **Risk: Exposure of Data Plane Traffic via Public Endpoints (High Severity)**
  + **Architecture Component:** Application MongoDB VPC Endpoints (Data Plane).
  + **Details:** The document mandates that all traffic must be private via VPC endpoints (see "Network Connectivity to MongoDB - Data Plane" section). If applications are misconfigured to use public endpoints instead of SkyTransit or VPC endpoints, sensitive data could be exposed over the public internet.
  + **Cause:** Lack of monitoring or enforcement to ensure all traffic uses private endpoints.
* **Risk: Overlapping Port Ranges in Multi-Region Deployments (Medium Severity)**
  + **Architecture Component:** MongoDB VPC Endpoints (Data Plane).
  + **Details:** The document specifies port ranges 1024-65535 for MongoDB PrivateLink/VPCE (see "Network Connectivity to MongoDB - Data Plane" section). In multi-region deployments, if port assignments overlap or are not properly managed, it could lead to connectivity issues or conflicts, forcing traffic to less secure routes.
  + **Cause:** Lack of automated port management for multi-region setups.

#### **Authentication**

* **Risk: Over-Privileged Access to MongoDB Atlas UI (Medium Severity)**
  + **Architecture Component:** MongoDB Atlas Control Plane (UI Access).
  + **Details:** The document requires Control/Management Plane access to the MongoDB Atlas UI to use MFA via GS SecurID (see "Authentication" section). If MFA isn’t consistently enforced or if too many users are granted access without following the principle of least privilege, unauthorized users could gain administrative access.
  + **Cause:** Weak access control practices or insufficient permission audits.
* **Risk: Weak API Key Management for Programmatic Access (Medium Severity)**
  + **Architecture Component:** MongoDB Atlas Control Plane (API Keys).
  + **Details:** The document assigns the ME Team to manage API keys for programmatic access (see "Authentication" section). If these keys aren’t rotated regularly or are stored insecurely, they could be compromised, allowing attackers to access the MongoDB Atlas Control Plane.
  + **Cause:** Lack of key rotation policies or secure storage mechanisms.

#### **Encryption**

* **Risk: Failure to Rotate Encryption Keys Periodically (High Severity)**
  + **Architecture Component:** MongoDB Control Plane (KMS Encryption Key).
  + **Details:** The document requires GS CMK/BYOK for data encryption at rest and periodic key rotation per Tech Risk guidelines (see "Encryption" section). If keys aren’t rotated as required, or if the rotation process fails, it could weaken data security, making it easier for attackers to decrypt data if a key is compromised.
  + **Cause:** Lack of automated key rotation processes or oversight by the Application Team.
* **Risk: Incomplete Client-Side Encryption Implementation (Medium Severity)**
  + **Architecture Component:** Application to MongoDB Data Plane.
  + **Details:** The document mentions client-side encryption as defined in the Extended Data Classification and Protection Model (see "Encryption" section). If applications don’t fully implement client-side encryption before sending data to MongoDB, sensitive data could be transmitted unencrypted, increasing the risk of interception.
  + **Cause:** Lack of developer training or enforcement of client-side encryption standards.

#### **Logging**

* **Risk: Inadequate Audit Logs for Administrative Actions (Medium Severity)**
  + **Architecture Component:** MongoDB Atlas Control Plane.
  + **Details:** The document states that Middleware Engineering enables database auditing (see "Logging" section). If audit logs don’t capture detailed administrative actions (e.g., who made changes, when, and what), it could hinder incident response and forensic analysis in case of a security breach.
  + **Cause:** Incomplete logging configurations or lack of integration with downstream systems like PAR and Surveillance.

#### **Monitoring**

* **Risk: Insufficient Monitoring of Application-Specific Metrics (Medium Severity)**
  + **Architecture Component:** MongoDB Instances (Data Plane).
  + **Details:** The document requires application-specific firm-approved monitoring (see "Monitoring" section). If monitoring doesn’t include key metrics like query performance, connection failures, or unauthorized access attempts, issues could go undetected, leading to downtime or security incidents.
  + **Cause:** Incomplete monitoring setup or lack of integration with GS monitoring tools.

#### **Project/User Management**

* **Risk: Lack of Granular Access Control for Project Users (Medium Severity)**
  + **Architecture Component:** MongoDB Atlas Control Plane (User Access).
  + **Details:** The document mentions managing project application control plane API keys (see "Project/User Management" section). If user access isn’t granularly controlled (e.g., assigning overly broad permissions), it could lead to unauthorized access to sensitive project settings or data.
  + **Cause:** Failure to enforce least privilege principles during user provisioning.

#### **Inventory/Data Classification**

* **Risk: Misclassification of Data Sensitivity Levels (Medium Severity)**
  + **Architecture Component:** MongoDB Instances (Data Plane).
  + **Details:** The document requires data classification per the firm-approved inventory (see "Inventory/Data Classification" section). If data sensitivity levels (e.g., Restricted, Highly Restricted) are misclassified, it could lead to inadequate protection measures, exposing sensitive data to breaches.
  + **Cause:** Lack of automated data classification tools or insufficient training for Application Teams.

#### **Multi-Region Deployment**

* **Risk: Inconsistent VPC Endpoint Configuration Across Regions (Medium Severity)**
  + **Architecture Component:** MongoDB VPC Endpoints (Data Plane).
  + **Details:** The document requires IAM roles for multi-region deployments (see "Multi-Region Deployment" section). If VPC endpoints aren’t consistently configured across regions, it could lead to connectivity issues or force applications to use less secure public endpoints, exposing traffic.
  + **Cause:** Manual configuration errors or lack of automation for multi-region VPC setups.

#### **Tenancy**

* **Risk: Shared Cluster Misuse in Production (Medium Severity)**
  + **Architecture Component:** MongoDB Instances (Data Plane).
  + **Details:** The document specifies dedicated Atlas clusters for production (M30 and above) and shared clusters for non-production (M10) (see "Tenancy" section). If production workloads are accidentally deployed on shared clusters (e.g., due to cost concerns), it could lead to performance degradation or data leakage between tenants.
  + **Cause:** Lack of governance to enforce dedicated clusters for production environments.

Let’s break down the "MongoDB Architecture: Tech Risk Connectivity and Usage Pattern" document into very simple terms, explaining every single point clearly and easily. This document is a guide for teams to use MongoDB (a cloud database) safely in a company. It splits tasks between the **Application Team** (who makes the app) and the **Middleware Engineering (ME) Team** (who supports the tech setup). Let’s go through each section and point in a beginner-friendly way.

### **What’s This Document For?**

MongoDB is like a big digital filing cabinet where apps store information, like customer names or orders. This document tells two teams how to set it up and use it safely so data doesn’t get stolen or lost. It was last updated on October 16, 2024, by someone named Kishor Pangaluru.

It covers:

* Setting up the database.
* Connecting to it securely.
* Protecting the data.
* Keeping track of who uses it.
* Watching for problems.
* Handling setups in multiple places.
* Choosing the right size for the database.

### **Section 1: Responsibilities (Who Does What?)**

This section lists tasks for the **Application Team** (app builders) and the **ME Team** (tech support). It also gives tips (called "Notes/Best Practices").

#### **1. Managing Resources (Setting Up MongoDB)**

* **Application Team**:
  + Create MongoDB projects (like folders for your app’s data) using a tool called the ME Portal (a website or app).
  + Set up things like:
    - **Clusters**: A group of servers that store your data.
    - **Connectivity**: How your app talks to MongoDB.
    - **System Users**: Accounts for the system to use MongoDB.
    - **Database Users**: Accounts for people or apps to access data.
    - **Roles**: Rules about who can do what (like read or write data).
    - **GS CMK for Encryption**: A special key to lock your data so it’s safe.
    - **Cluster Backup Policy**: Rules to back up your data so it’s not lost.
* **ME Team**:
  + Set up the big MongoDB system (called Atlas organizations), which is like the main library that holds all the projects.
  + Turn on a tool called DACT (Data Access Control Tool) to make sure only the right people can access the database.
  + Give the Application Team an easy way to manage their projects through the ME Portal.
* **Tips**:
  + No tips listed here.

#### **2. Using Code to Set Up MongoDB (IaC/SDLC)**

* **Application Team**:
  + Use tools called SkyFoundry or Terraform to set up MongoDB (these tools let you set things up with code instead of clicking buttons).
  + Create database users with a tool called CloudFormation Templates (CFT). This is highlighted because it’s important.
  + Make sure users only get the access they need (e.g., don’t let someone who only needs to read data also delete it).
* **ME Team**:
  + Give the Application Team a way to manage MongoDB settings through code, like giving them a toolbox to build things.
* **Tips**:
  + No tips listed here.

#### **3. Connecting to MongoDB (Network Setup)**

* **Application Team**:
  + Make sure your app talks to MongoDB privately (not over the public internet, so it’s safer).
  + Use special pathways called VPC endpoints (like private tunnels) to connect. You can set this up with tools like IaC/SDLC or SkyFoundry/Terraform.
  + Tell the system which apps are allowed to talk to MongoDB (like making a list of approved friends).
  + Use a rule called ConnMan to let your app send messages from the company network (GSINet) to MongoDB’s network (SkyTransit). This rule depends on your project and region (like the US or Europe).
  + For testing (non-production), you need special permission to access MongoDB. For real use (production), no one outside the company can access it.
* **ME Team**:
  + Set up private tunnels (VPC endpoints) for their own tools, not for the app.
  + Automatically find private tunnels made by the Application Team and set up matching tunnels in the company network (GSINet). These tunnels are for tools like DACTLite or Privacy Inventory Management, not for the app to use.
* **Tips**:
  + MongoDB uses ports 1024–65535 (think of ports like door numbers for sending messages).
  + These ports are approved and needed for the design to keep things private.
  + Using private tunnels saves money because MongoDB charges per region (like per country), not per server.
  + If you remove a MongoDB server, the port numbers might change, which can cause problems. Be careful when changing setups.

#### **4. Logging In Securely (Authentication)**

* **Application Team**:
  + **For People (Data Plane)**:
    - Use a tool called DACT to log in with a temporary username and password (like a one-time ticket).
    - To use the MongoDB control panel (a website to manage MongoDB), log in with your company account (GS Single Sign-On) and extra security (MFA, like a code on your phone).
  + **For Apps (Data Plane)**:
    - Let your app log in using special roles (IAM roles), which are like ID cards for apps.
    - Only give the app the access it needs (e.g., if it only needs to read data, don’t let it delete data).
  + **For Control (Control/Management Plane)**:
    - Use special keys to control MongoDB through code (like a remote control for MongoDB).
    - Manage these keys for your app.
* **ME Team**:
  + Store these keys in a safe place called Secrets Manager (like a locked box).
  + Make sure only approved devices can use these keys (like checking IDs before letting someone in).
  + Turn on the company login (Single Sign-On) for the MongoDB control panel.
* **Tips**:
  + No tips listed here.

#### **5. Protecting Data (Encryption)**

* **Application Team**:
  + Use a company key (GS CMK) or your own key (BYOK) to lock data when it’s sitting in MongoDB (so no one can steal it).
  + Change this key regularly to keep it safe (like changing your password).
  + When data is sent to MongoDB, it’s automatically locked with something called TLS/SSL (like putting it in a sealed envelope).
  + If the data is super sensitive (like credit card numbers), lock it before sending it to MongoDB (called client-side encryption).
* **ME Team**:
  + No tasks here.
* **Tips**:
  + You must use a key (GS CMK/BYOK) to lock data in MongoDB—it’s a rule.
  + MongoDB automatically locks data when it’s sent (TLS/SSL is turned on by default). You can read more about it on MongoDB’s website.

#### **6. Keeping Logs (Logging)**

* **Application Team**:
  + Make sure your app keeps a record of what it does with MongoDB (like a diary of actions).
* **ME Team**:
  + For real use (production), turn on a system to track what happens in MongoDB and send these records to the security team (called PAR and Surveillance).
* **Tips**:
  + No tips listed here.

#### **7. Watching for Problems (Monitoring)**

* **Application Team**:
  + Set up alerts to catch problems with your app’s use of MongoDB (like getting a text if something breaks).
* **ME Team**:
  + No tasks here.
* **Tips**:
  + No tips listed here.

#### **8. Keeping Track of Data (Inventory/Data Classification)**

* **Application Team**:
  + Make a list of all the data you store in MongoDB (like names, emails) and label it (e.g., “sensitive” or “not sensitive”) in a company system.
* **ME Team**:
  + No tasks here.
* **Tips**:
  + No tips listed here.

#### **9. Using MongoDB in Multiple Places (Multi-Region Deployment)**

* **Application Team**:
  + If your app uses MongoDB in different places (like the US and Europe), set up special access rules (IAM roles) for each place.
* **ME Team**:
  + Make sure the app follows rules for using MongoDB in different places.
* **Tips**:
  + No tips listed here.

#### **10. Choosing the Right Size (Tenancy)**

* **Application Team**:
  + For testing (non-production), use a smaller MongoDB setup (called M10 or bigger).
  + For real use (production), use a bigger setup (called M30 or bigger).
* **ME Team**:
  + Use a dedicated setup (like a private room) for real use, but a shared setup (like a shared room) for testing.
* **Tips**:
  + No tips listed here.

### **Section 2: How It All Connects (Diagram)**

This picture shows how your app talks to MongoDB safely:

* Your app starts in the company network (called GSINet).
* It connects to MongoDB through secure tunnels (VPC Endpoints/Private Links) using ports 1024–65535 (like special door numbers).
* The app logs in using a tool called Skybroker, which gives it a secure ticket (Auth Payload/OAuth Token) from a system called PingFederate.
* All messages are locked with TLS/HTTPS (like sending a letter in a sealed envelope).
* MongoDB has its own space (MongoDB VPC) with its servers (MongoDB Instances).
* A control system (MongoDB Control Plane) manages the keys to lock data (KMS).
* Firewalls (like security guards) make sure only approved messages get through.
* The company login (GSSSO) helps people access MongoDB safely.

### **Section 3: What Could Go Wrong? (Risks)**

This section lists problems that might happen and how serious they are (Low, Medium, or Info):

1. **No Good List of Stuff (Medium)**:
   * The company doesn’t have a good list of all apps, databases, or sensitive data, so it’s hard to keep track.
2. **Same ID for Different Things (Medium)**:
   * Using the same ID for different setups can cause mix-ups (like two people sharing the same library card).
3. **Mistakes by People (Medium)**:
   * Managing users by hand (instead of with a system) can lead to mistakes (like giving someone the wrong access).
4. **Too Many Permissions (Medium)**:
   * Some people get too much power (like being able to delete everything when they only need to read).
   * One person shouldn’t have power over multiple areas (like being the boss of two different teams).
5. **Not Tracking Users (Info)**:
   * The company doesn’t know who has access to what (like not knowing who has keys to the filing cabinet).
6. **Too Much Access (Low)**:
   * A system called Security Sync gives some users full access when they don’t need it (like giving someone a master key for no reason).
7. **Using Powerful Accounts for Small Tasks (Medium)**:
   * Using a high-power account for simple jobs (like using a big truck to deliver a small package).
8. **No Good Logs (Medium)**:
   * The company doesn’t keep detailed records of what happens in MongoDB (like not having a guest book).
   * These records aren’t stored in the company’s system (GS environment).

### **Section 4: Key Rules to Follow (Tech Risk Patterns)**

This section sums up the main rules for using MongoDB safely.

#### **Application Team Rules:**

* Set up MongoDB projects and manage things like clusters, users, and keys.
* Use tools like SkyFoundry/Terraform to set up MongoDB with code.
* Create database users with CloudFormation Templates (CFT).
* Only give users the access they need (not too much).
* Connect to MongoDB privately using secure tunnels (VPC endpoints).
* Use a rule (ConnMan) to let your app talk to MongoDB safely.
* For testing, get permission to access MongoDB. For real use, no outside access is allowed.
* Use DACT for people to log in with temporary passwords.
* Use the company login (GS Single Sign-On) with extra security (MFA) for the MongoDB control panel.
* Let your app log in with special roles (IAM roles).
* Manage keys to control MongoDB through code.
* Lock data with a company key (GS CMK) or your own key (BYOK).
* Change the key regularly.
* Data is automatically locked when sent (TLS/SSL).
* Lock sensitive data before sending it to MongoDB.
* Keep a record of what your app does with MongoDB.
* Set up alerts to catch problems.
* List all your data in a company system and label it (e.g., “sensitive”).
* Set up special access rules (IAM roles) for using MongoDB in different places.
* Use a small setup (M10) for testing and a big setup (M30) for real use.

#### **ME Team Rules:**

* Set up the big MongoDB system (Atlas organizations).
* Turn on DACT for secure access.
* Give the Application Team tools to manage projects easily.
* Provide a way to manage MongoDB settings through code.
* Set up private tunnels (VPC endpoints) for their own tools (not for the app).
* Store keys in a safe place (Secrets Manager).
* Make sure only approved devices can use these keys.
* Turn on the company login (Single Sign-On) for the MongoDB control panel.
* Keep logs of what happens in MongoDB for real use and send them to the security team.
* Use a dedicated setup for real use and a shared setup for testing.

#### **Tips:**

* Ports 1024–65535 are used for secure connections.
* These ports are approved and save money by charging per region.
* Be careful when changing setups because port numbers might change.
* You must use a key (GS CMK/BYOK) to lock data.
* MongoDB automatically locks data when it’s sent (TLS/SSL).

### **Why This Matters**

This guide helps keep MongoDB safe by:

* Making sure only the right people and apps can use it.
* Locking data so it’s not stolen.
* Keeping records and watching for problems.
* Following company rules to avoid mistakes.