**Use Case Document: Global Monitoring Model**

# Executive Summary

Provide a brief overview of the purpose of the Global MongoDB Monitoring Model, its significance, and the expected outcomes.

# Objectives

The specific objectives of the POC, including:

* Implementing a robust monitoring model for MongoDB Atlas clusters.
* Evaluating the performance, scalability, and efficiency of MongoDB Atlas in a global context.
* Establishing effective alerting mechanisms and response strategies for monitoring events.
* The objective of this use case is to automate the extraction of metrics from MongoDB Atlas using its API, convert the output to JSON, format it into a tabular form, and then load the data into a SQL Server database.

# Scope

As part of this implementation, we need to setup new Organization, Project and cluster with test data loaded into collections. We will use Atlas API keys to authenticate and authorize clusters to retrieve monitoring data. Atlas API call will be used to pull monitoring metrics and these metrics can be used in assessing the scalability and performance of the databases and infrastructure.

# Infrastructure Requirements

* Atlas Organization
* Project Creation
* Deploy Cluster
* Create Organization API keys and secret with role-based access
* Create database and collections
* Load Test data to generate monitoring metrics
* PowerShell and Required Modules

# Monitoring Model

## Key Metrics

Returns disk, partition, or host measurements per process for the specified host for the specified project. Returned value can be one of the following:

* Throughput of I/O operations for the disk partition used for the MongoDB process
* Percentage of time during which requests the partition issued and serviced
* Latency per operation type of the disk partition used for the MongoDB process
* Amount of free and used disk space on the disk partition used for the MongoDB process
* Measurements for the host, such as CPU usage or number of I/O operations

## MongoDB Atlas Monitoring Tools

The Atlas Administration API follows the principles of the REST architectural style to expose a number of internal resources which enable programmatic access to Atlas's features and it provides all the performance metrics with API calls.

# Implementation Steps:

## Step 1: Install Required PowerShell Modules

# Install required modules

Install-Module -Name dbatools

Install-Module -Name MongoDB

## Step 2: Authenticate with Atlas API

# Replace 'API\_KEY' and 'API\_SECRET' with your actual API key and secret

$apiKey = 'API\_KEY'

$apiSecret = 'API\_SECRET'

# Authenticate with Atlas API

$authHeader = @{ 'Authorization' = "Digest ${apiKey}:${apiSecret}" }

## Step 3: Pull Metrics from Atlas API and Convert to JSON

# Replace 'CLUSTER\_ID' with your MongoDB Atlas cluster ID

$clusterId = 'CLUSTER\_ID'

$baseUrl = "https://cloud.mongodb.com/api/atlas/v1.0/groups/${clusterId}/processes"

# Pull metrics from Atlas API

$response = Invoke-RestMethod -Uri $baseUrl -Headers $authHeader -Method Get

# Convert to JSON

$jsonMetrics = $response | ConvertTo-Json

## Step 4: Format JSON Metrics into Tabular Form

# Convert JSON to PowerShell objects

$objects = $jsonMetrics | ConvertFrom-Json

# Format the objects into a table

$table = $objects | Format-Table -AutoSize

## Step 5: Load Metrics into SQL Server with Service Account Authentication

# Replace 'SQL\_SERVER', 'DATABASE', 'TABLE' with your SQL Server details

$sqlServer = 'SQL\_SERVER'

$database = 'DATABASE'

$tableName = 'TABLE'

$serviceAccount = 'DOMAIN\ServiceAccount'

$password = 'ServiceAccountPassword'

# Create a secure string for the password

$securePassword = ConvertTo-SecureString $password -AsPlainText -Force

# Use dbatools to insert data into SQL Server with service account authentication

$table | Write-DbaDbTableData -SqlInstance $sqlServer -Database $database -Table $tableName -SqlCredential (New-Object PSCredential -ArgumentList $serviceAccount, $securePassword)