66% of organizations use between 2-5 monitoring or observability tools.

24% of organizations have breached a contractual service level agreement in the last 12 months

Catchpoint SRE Report 2024

AWS Observability as Code Leveraging Datadog for Advanced Platform Engineering

Indika Wimalasuriya

Platform Engineering - 2024

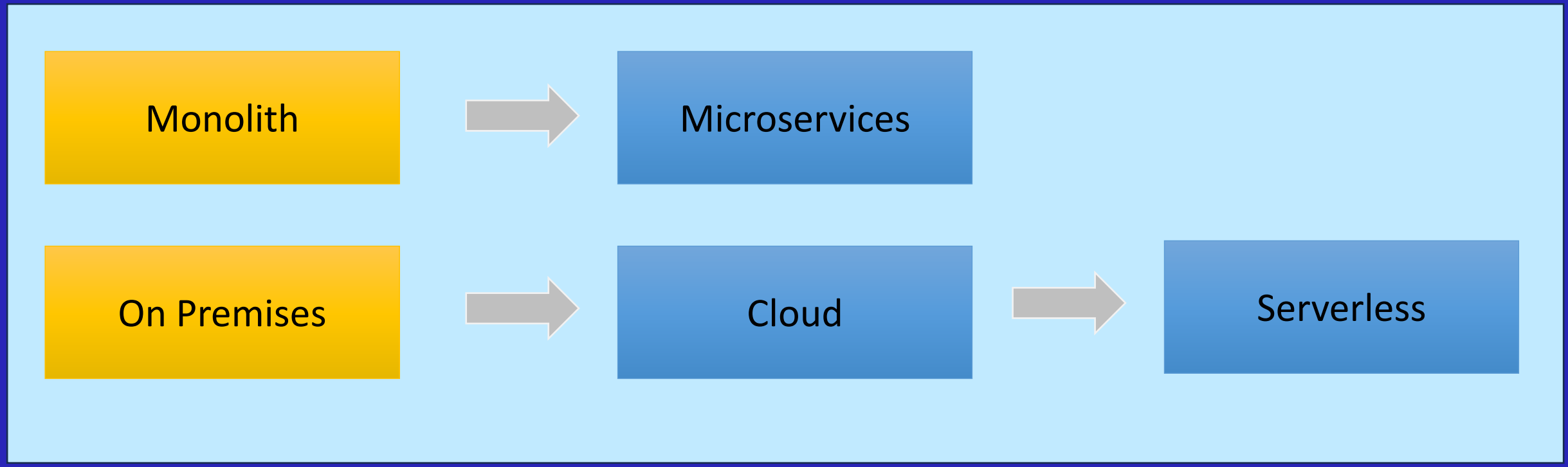
Agenda

- Understanding Observability in Modern Platforms
- Role of Observability in Platform Engineering
- Overview of Datadog
- Implementing Observability as Code
- Real-World Implementation
- Best Practices and Pitfalls to Avoid

Quick Intro about myself

- Resides in Colombo, Sri Lanka, with my daughter and wife.
- Reliability Engineering Advocate, Solution Architect (specializing in SRE, Observability, AIOps, & GenAI).
- Employed at Virtusa, overseeing technical delivery and capability development.
- Passionate Technical Trainer.
- Energetic Technical Blogger.
- AWS Community Builder - Cloud Operations.
- Ambassador at DevOps Institute (PeopleCert).

Managing Ever-Growing Complexity in Distributed Systems



Expansion of Data Sources

Surge in Data Volume

Exponential Rise in Failure Scenarios

Understanding Observability in Modern Platforms

Monitoring → Observability

Distributed System Complexity	Dynamic and Elastic Nature	Container Orchestration
Continuous Deployment and Integration	Service Mesh and API gateways	Increased Complexity of Cloud Services
Scalability Challenges	Quick Detection and Resolution of Issues	End to End Visibility

Observability

- 📝 Logs
- 📊 Metrics
- 🔍 Tracing
- 🚨 Alarms
- 📈 Dashboards
- 🐦 Canaries
- 🌐 Real User Monitoring
- 🖥️ Infrastructure Monitoring
- 🌍 Network Monitoring
- 🔒 Security Monitoring
- 💰 Cost Optimization

Role of Observability in Platform Engineering

Unified Observability Strategy

- Consistency Across Teams
- Centralized Monitoring

Efficient Troubleshooting and Incident Response

- Cross-Team Collaboration
- Faster Root Cause Analysis

Scalability and Reliability

- Proactive Monitoring
- Capacity Planning

Enhanced Developer Experience

- Self-Service Observability
- Standardized Dashboards

Security and Compliance

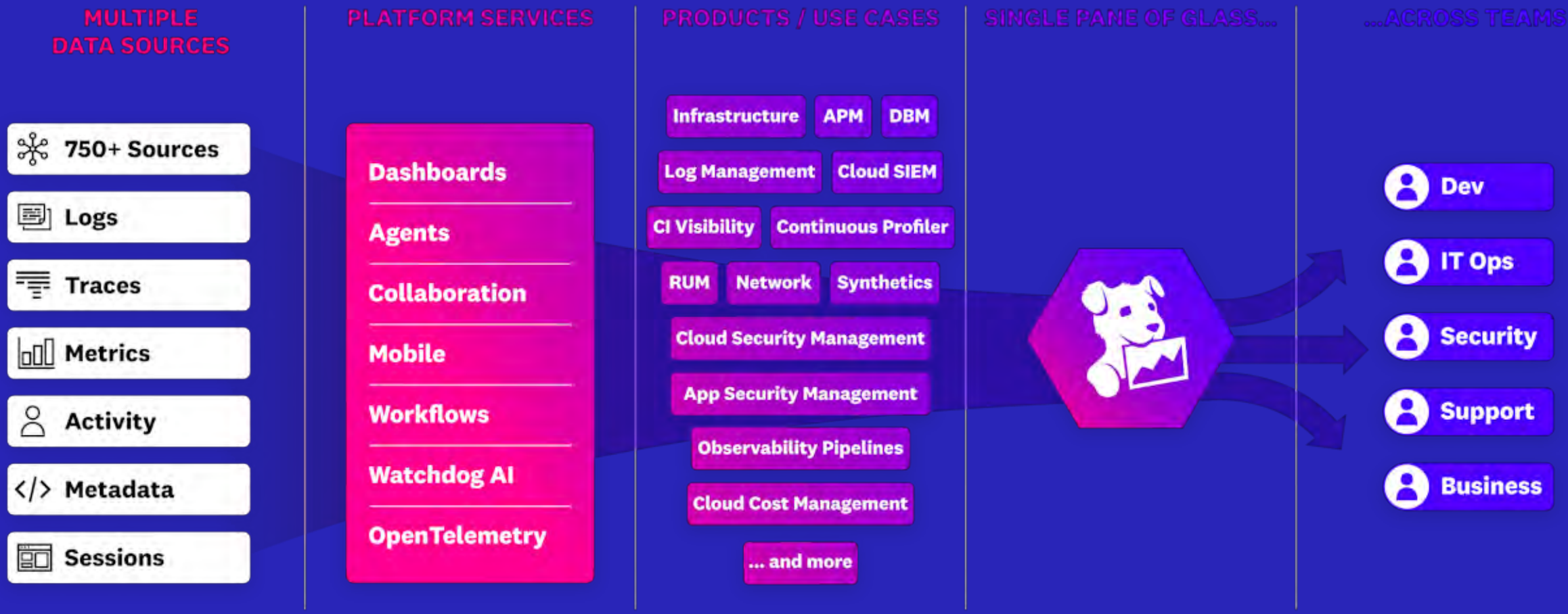
- Unified Security Monitoring
- Audit and Reporting

Cost Efficiency

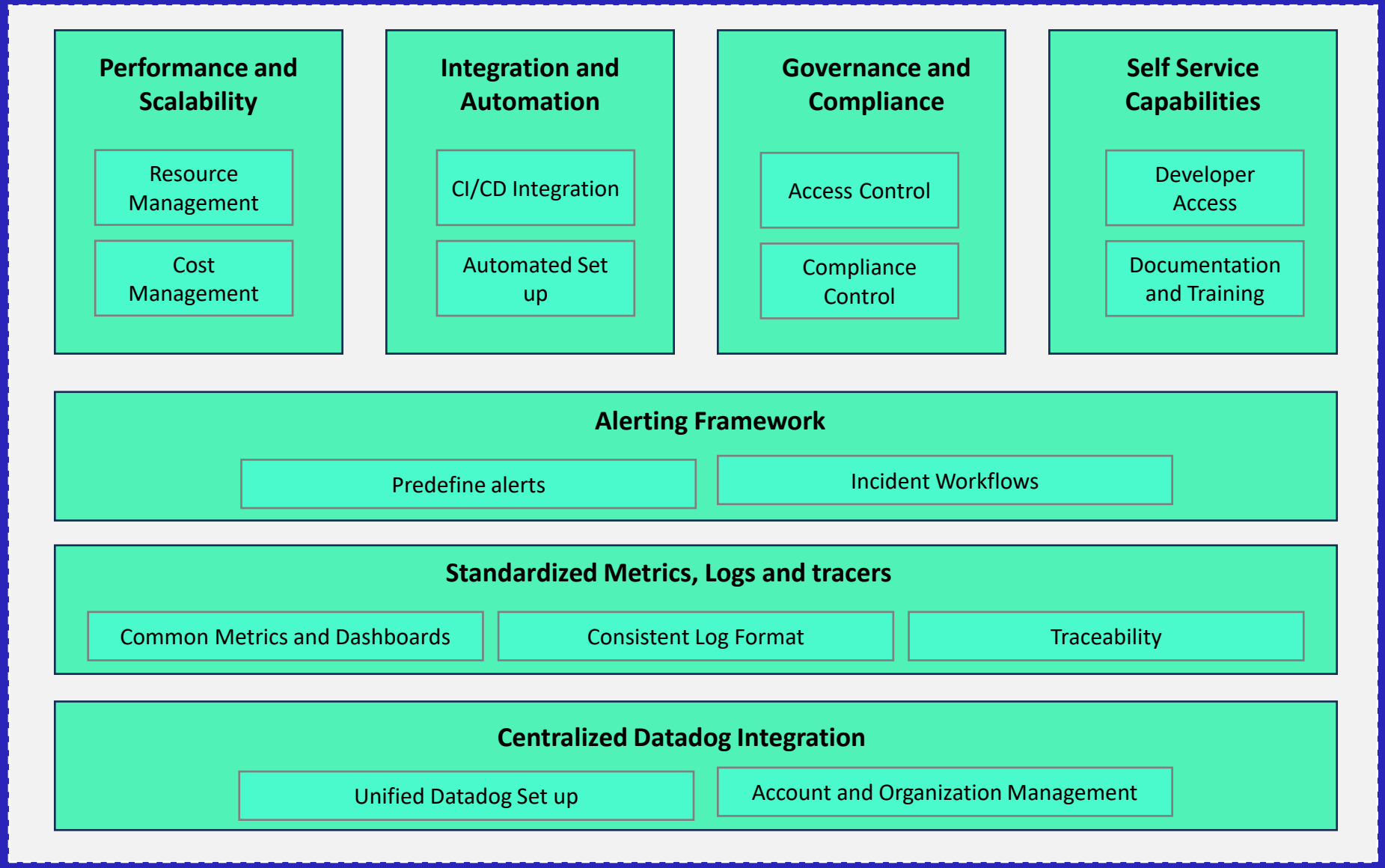
- Tool Consolidation
- Optimized Resource Usage

Overview of Datadog

DATADOG



Building a Unified Platform Engineering Framework: Enhancing AWS Observability with Datadog



Observability as Code

Treat Configurations as Code

Metrics, logs, and tracing setups are managed as code artifacts.

Versioning and Management

Use version control systems (e.g., Git) for tracking changes and collaboration.

Automation and CI/CD

Automate deployment and updates through CI/CD pipelines.
Integrate with Infrastructure as Code (IaC) tools for standardized provisioning.

Consistency and Standardization

Implement standardized templates and modules for observability configurations.
Apply consistent settings across different environments.

Monitoring and Maintenance

Conduct reviews and validation of observability configurations.
Implement testing frameworks to ensure proper functionality.

Documentation and Collaboration

Document observability code and provide guidelines.
Enhance collaboration with shared observability practices and code.

Benefits of Observability as Code

🔍 Improved Accuracy

📊 Enhanced Visibility

⚡ Faster Deployment

🔄 Consistency Across Environments

🤝 Better Collaboration

📈 Continuous Improvement

Observability as Code Tools

Real World Implementations

- Datadog Integration Automation
- Log Pipeline Setup
- Enable APM (Application Performance Monitoring)
- Enable Common Metrics and Dashboards
- Create Synthetic Monitors
- Create Alerts
- Account Creation and Access Control
- CI/CD Pipeline Integration
- Distributed Tracing Setup
- Service Level Objectives (SLOs) Configuration
- Automated Incident Management Integration
- Compliance and Governance Controls

Workflow of Observability as Code

Step	Description
Define Observability Configuration	Identify and create configuration files for key observability components such as metrics, logs, tracing setups, dashboards, and alerts.
Version Control	Store configuration files in a version control system (e.g., Git), and manage changes through branching and merging strategies.
Test Configurations	Test the observability configurations locally or in a staging environment; validate syntax, logic, and performance.
Continuous Integration (CI)	Integrate configurations with CI pipelines, automatically running tests on changes to ensure they pass before deployment.
Automate Deployment	Use IaC tools (e.g., Terraform, Ansible) to automate the deployment of observability configurations across all environments.
Monitoring and Validation	Monitor the deployment process and validate that observability components are functioning correctly and are properly integrated.
Review and Continuous Improvement	Regularly review and update observability configurations based on feedback and evolving business needs.
Maintenance and Compliance	Ensure ongoing maintenance of configurations, including updates and compliance checks to meet organizational and regulatory standards.

Terraform Folder Structure

```
observability-as-code/
│
├── deployment/
│   ├── main.tf                    # Main entry point for Terraform
│   ├── providers.tf               # Provider configurations (e.g., Datadog)
│   ├── variables.tf               # Variable definitions
│   ├── versions.tf                # Terraform version and provider version constraints
│   ├── vars/
│   │   ├── monitors.yml           # YAML file for monitor configurations
│   │   ├── slo.yml                # YAML file for SLO configurations
│   │   ├── dashboard.yml          # YAML file for dashboard configurations
│   │   └── maintenance_window.yml # YAML file for maintenance window configurations
│
├── modules/
│   ├── monitors/
│   │   ├── main.tf                # Monitor resource configurations
│   │   ├── variables.tf           # Variables specific to monitors
│   │   └── outputs.tf             # Outputs related to monitors
│   │
│   ├── slo/
│   │   ├── main.tf                # SLO resource configurations
│   │   ├── variables.tf           # Variables specific to SLOs
│   │   └── outputs.tf             # Outputs related to SLOs
│   │
│   ├── dashboard/
│   │   ├── main.tf                # Dashboard resource configurations
│   │   ├── variables.tf           # Variables specific to dashboards
│   │   └── outputs.tf             # Outputs related to dashboards
│   │
│   └── maintenance_window/
│       ├── main.tf                # Maintenance window resource configurations
│       ├── variables.tf           # Variables specific to maintenance windows
│       └── outputs.tf             # Outputs related to maintenance windows
```

Datadog Integration Automation

```
# Datadog provider configuration in Terraform
provider "datadog" {
  api_key = var.datadog_api_key  # Datadog API key, securely stored in Terraform variables
  app_key = var.datadog_app_key  # Datadog Application key, also stored in Terraform
variables

  # Optional configuration settings:
  api_url = "https://api.datadoghq.com"  # API URL, default is Datadog US region, adjust if
needed
  validate = true  # Validate the provider configuration; set to false to skip validation checks
}

# Variable definitions to securely pass API and application keys
variable "datadog_api_key" {
  description = "Datadog API key, used to authenticate API requests."
  type        = string
  sensitive   = true  # Marked as sensitive to avoid displaying in logs or outputs
}

variable "datadog_app_key" {
  description = "Datadog Application key, used for accessing Datadog API."
  type        = string
  sensitive   = true  # Marked as sensitive to protect the key
}

# Datadog Agent installation for Linux-based systems
resource "null_resource" "install_datadog_agent" {
  # Provisioner to run commands for installing the Datadog Agent
  provisioner "remote-exec" {
    inline = [
      "DD_AGENT_MAJOR_VERSION=7 DD_API_KEY=${var.datadog_api_key}
DD_SITE='datadoghq.com' bash -c \"$(curl -L https://s3.amazonaws.com/dd-
agent/scripts/install_script.sh)\"",
    ]
```

```
  # Connection details for remote execution
    connection {
      type     = "ssh"
      user     = "ubuntu"  # Adjust based on your Linux distribution
      host     = var.instance_ip  # IP address of the target instance
      private_key = file(var.ssh_private_key_path)  # Path to SSH private key for secure
connection
    }
  }

  # Dependency on the instance to ensure the agent is installed after the instance is created
  depends_on = [
    aws_instance.my_instance  # Replace with your instance resource name
  ]
}

# Variable definition for the instance IP address
variable "instance_ip" {
  description = "IP address of the instance where the Datadog Agent will be installed."
  type        = string
}

# Variable definition for the SSH private key path
variable "ssh_private_key_path" {
  description = "Path to the SSH private key used for connecting to the instance."
  type        = string
}
```

Log pipeline set up

```
# Datadog Logs Pipeline Configuration
resource "datadog_logs_pipeline" "pipeline" {
  name = "my-log-pipeline"  # Descriptive name for the log pipeline

  # Filter block to specify the logs that should be processed by this pipeline
  filter {
    query = "status:error"  # Filter to include only logs where the status is 'error'
  }

  # Additional processors can be added to the pipeline to modify or enrich logs
  processor {
    grok {
      samples = [
        "127.0.0.1 - john.doe [12/Dec/2023:15:03:14 +0000] \"GET /index.html HTTP/1.1\" 200
5123",  # Example log sample for pattern matching
      ]
      support_rules = false  # Disable support rules for simple pattern extraction
      grok {
        match_rules = {
          "client_ip"   = "%{IP:client_ip}",  # Extract client IP address from the log
          "timestamp"   = "%{HTTPDATE:timestamp}",  # Extract timestamp
          "method"      = "%{WORD:method}",  # Extract HTTP method
          "url"         = "%{DATA:url}",  # Extract URL
          "status_code" = "%{NUMBER:status_code:int}",  # Extract HTTP status code as integer
        }
        support_rules = true  # Enable support rules for complex pattern extraction
      }
    }
  }

  # The sort parameter defines the order in which logs are processed in the pipeline
  sort = 1  # Lower numbers have higher priority; this pipeline processes logs first
```

```
  # Use is_enabled to control whether the pipeline is active
  is_enabled = true  # Pipeline is active and processes logs

  # Sample logs to verify the pipeline's behavior and configuration
  sample_logs = [
    "{\"status\": \"error\", \"message\": \"An error occurred.\"}",
    "{\"status\": \"ok\", \"message\": \"All systems operational.\"}",
  ]
}
```

Create Synthetic Monitors

```
terraform {
  required_version = ">= 0.15.5"  # Minimum Terraform version required
  required_providers {
    datadog = {
      source  = "DataDog/datadog" # Source for Datadog provider
      version = ">= 3.0.0"       # Minimum version of Datadog provider required
    }
  }
}

provider "aws" {
  region = "us-east-1"          # AWS region where resources will be deployed
}

provider "datadog" {
  api_key = "0896a2a9199865e97bc08f411106ebac"  # Datadog API key
  app_key = "a7c980c57b02cdde488b8e683185aa928ae8f7ef"  # Datadog Application key
  api_url = "https://us5.datadoghq.com/"       # Datadog API URL, specific to your region
}

resource "datadog_synthetics_test" "test_uptime" {
  name     = "Synthetics Test"      # Name of the synthetic test
  type     = "api"                  # Type of the test, set to 'api'
  subtype  = "http"                 # Subtype of the test, set to 'http'
  status   = "live"                 # Status of the test, set to 'live'

  message  = "Notify "              # Notification message
  locations = ["aws:eu-central-1", "aws:us-east-1", "aws:ap-southeast-1"]  # Locations to run
the test from

  request_definition {
    method = "GET"                  # HTTP method to be used for the request
    url    = "http://54.160.164.216/datadog_monitor.html"  # The URL that will be tested
  }
```

```
  assertion {
    type     = "statusCode"        # Type of assertion, here it's checking the status code
    operator = "is"                # Operator used in the assertion (e.g., is, contains)
    target   = "302"               # Expected value of the assertion (HTTP status code 302)
  }

  options_list {
    tick_every = 200               # Frequency of the test in seconds
    retry {
      count    = 2                 # Number of retries in case of failure
      interval = 300               # Interval between retries in seconds
    }
    monitor_options {
      renotify_interval = 120      # Interval in minutes for renotification if the issue persists
    }
  }
}
```

Create Alerts

```
resource "datadog_monitor" "alert" {
  name  = "High CPU Alert"         # Name of the Datadog monitor, describing the alert's purpose

  type  = "metric alert"           # Type of the monitor, specifying that this is a 'metric alert'

  # Query that defines the condition for triggering the alert.
  # This example checks if the average CPU usage of the system, over the last 5 minutes, exceeds 80%.
  query = "avg(last_5m):avg:system.cpu.user{*} > 80"

  message = "High CPU usage detected. Please investigate."  # Notification message sent when the alert is triggered

  tags = ["team:devops", "env:production"]  # Tags to categorize and filter the monitor, e.g., by team or environment

  # Threshold options for the alert
  thresholds {
    critical = 80     # Critical threshold, alert is triggered when CPU usage exceeds 80%
    warning  = 70     # Warning threshold, sends a warning when CPU usage exceeds 70%
  }

  # Options to customize the behavior of the monitor
  monitor_options {
    notify_no_data  = false          # Set to true if you want to be notified when data is missing
    no_data_timeframe = 10           # The time period (in minutes) to wait before triggering no data alerts
    renotify_interval = 60           # Time interval (in minutes) for renotification if the alert remains in a triggered state
    escalation_message = "CPU usage has been above 80% for more than 5 minutes. Immediate action required."  # Escalation message for critical alerts
  }
```

```
  # Notification settings to define who will receive the alerts
  notify_audit = true              # Notify when an alert is resolved, muted, or unmuted
  notification_channel = "@slack-channel"  # Slack channel or email list to notify when the alert is triggered

  # Defines the schedule for silencing alerts (e.g., during maintenance windows)
  silenced {
    start = "2023-09-01T00:00:00Z"   # Start time of the silence period in UTC
    end   = "2023-09-01T04:00:00Z"   # End time of the silence period in UTC
  }
}
```

Enable Common Metrics and Dashboards

```
resource "datadog_dashboard" "common_dashboard" {
  title      = "Common Metrics Dashboard"  # The title of the dashboard, indicating its purpose or focus
  layout_type = "ordered"                  # Specifies that the widgets on the dashboard should follow an ordered
layout

  # First widget group within the dashboard, used to logically group related widgets
  widget {
    group_definition {
      title      = "Overview"             # Title for the group, indicating that this section provides an overview of key
metrics
      layout_type = "ordered"              # Layout type for the group, ensuring the widgets are arranged in a
specified order

      # Timeseries widget definition, which visualizes metric data over time
      widget {
        timeseries_definition {
          title = "CPU Usage"              # Title for the timeseries widget, focusing on CPU usage metrics

          # Configuration for the timeseries widget, allowing for detailed customization
          requests {
            q = "avg:system.cpu.user{*}"   # Query to display the average CPU usage across all systems
            display_type = "line"          # Display type set to 'line', showing data trends over time
            style {
              palette = "cool"             # Color palette for the line, ensuring the visualization is easily distinguishable
              line_type = "solid"          # Type of line to be used in the graph, set as solid for clarity
              line_width = "normal"        # Width of the line, ensuring readability without overwhelming the graph
            }
          }
  # Customizing the appearance and behavior of the widget
          yaxis {
            scale = "linear"               # Y-axis scale set to linear for straightforward interpretation of values
min = "0"                                  # Minimum value for the Y-axis, ensuring that the graph starts at 0
            max = "100"                    # Maximum value for the Y-axis, matching common CPU usage percentage
limits
            include_zero = true            # Ensures that the Y-axis always includes zero, providing a full context for
the graph
            label = "CPU Usage (%)"        # Label for the Y-axis, clearly indicating the metric being displayed
          }
```

```
  }
      }

      # Additional widgets can be added here following the same pattern
      # This can include different metrics, visualizations, or groupings based on specific monitoring needs
    }
  }

  # Additional widget groups can be defined below, allowing for complex and comprehensive dashboards
  # Each group can have its own title, layout, and widget configurations
  widget {
    group_definition {
      title      = "Memory Usage"          # Example of a second widget group focusing on memory metrics
      layout_type = "ordered"              # Ensures widgets are arranged consistently within this group

      widget {
        timeseries_definition {
          title = "Memory Utilization"     # Title for the memory utilization widget
          requests {
            q = "avg:system.mem.used{*}"   # Query to show average memory utilization
            display_type = "area"          # Display type set to 'area' for showing stacked data
          }
          yaxis {
            label = "Memory Used (GB)"     # Label for the Y-axis, indicating memory usage in gigabytes
          }
        }
      }
    }
  }
}
```

Enable APM

```
resource "datadog_monitor" "apm_enabled" {
  name  = "APM Monitoring"  # The name of the monitor, indicating it is focused on APM
  type  = "apm"             # Specifies the monitor type as 'apm', which is used for tracking application performance

  # Query to monitor the average request duration of Flask requests, grouped by service
  query = "avg(last_5m):avg:trace.flask.request.duration{*} by {service}"

  # Notification section (optional), which specifies who to alert and how to notify them when the conditions are met
  message = "Average request duration for Flask services has exceeded the threshold. Please investigate the performance issue."  # The message that will be sent when the alert is triggered
  notify_no_data = false  # Specifies whether notifications should be sent if no data is received within the time frame
  notify_audit = false    # Determines whether changes to the monitor are logged

  # Options to customize how the monitor behaves, including alerting thresholds and evaluation timeframes
  options {
    thresholds {
      critical = 1000    # Threshold for triggering a critical alert (in milliseconds)
      warning  = 500     # Threshold for triggering a warning (in milliseconds)
    }
    evaluation_delay = 120  # Delay (in seconds) before evaluating the monitor to avoid false positives
    new_host_delay   = 300  # Delay (in seconds) before applying the monitor to newly discovered hosts
    include_tags     = true # Whether to include tags in the alert
    require_full_window = true  # Ensure the entire time window is used before triggering alerts
    notify_no_data   = true     # Send notifications when no data is received
  }
```

```
# Tags help to filter and manage the monitors within Datadog
  tags = [
    "env:production",       # Environment tag, indicating this monitor is for production
    "team:backend",         # Team responsible for this monitor
    "service:flask-requests"  # Specific service being monitored
  ]

  # Additional optional fields can be added to further customize the monitor, such as renotify intervals, escalation policies, etc.
}
```

Account creation and Access Control

```
# Datadog Role Configuration
resource "datadog_role" "admin_role" {
  name      = "Admin Role"  # Descriptive name for the role, such as 'Admin Role'

  # Permissions assigned to the role, defining the actions users with this role can perform
  permission = [
    "dashboards_read",  # Allows reading/viewing dashboards
    "dashboards_write",  # Allows creating, modifying, and deleting dashboards
    "monitors_read",  # Allows reading/viewing monitors
    "monitors_write",  # Allows creating, modifying, and deleting monitors
    "logs_read",  # Allows reading/viewing logs
    "logs_write",  # Allows creating, modifying, and deleting log configurations
    "synthetics_read",  # Allows reading/viewing synthetic tests
    "synthetics_write",  # Allows creating, modifying, and deleting synthetic tests
    "apm_read",  # Allows reading/viewing APM traces and metrics
    "apm_write",  # Allows creating, modifying, and deleting APM configurations
    "users_read",  # Allows viewing users and roles
    "users_write",  # Allows managing users and roles
    "account_manage",  # Allows managing account-level settings and billing
  ]

  # Scopes restrict the role's permissions to specific environments, teams, or services
  scopes = [
    "env:production",  # Restrict access to production environment resources
    "team:devops",  # Restrict access to resources tagged with 'team:devops'
  ]

  # Description of the role to provide additional context
  description = "Admin role with full access to dashboards, monitors, logs, and user
management across the production environment and DevOps team."  # Role description for
clarity

  # Ensure the role is active and ready for use
  is_enabled = true  # The role is active and can be assigned to users
}
```

CI/CD pipeline Integration

```
# Datadog CI/CD Integration Configuration
resource "datadog_integration" "ci_cd" {
  name = "CI/CD Integration"  # Descriptive name for the CI/CD integration, such as 'CI/CD Integration'

  # Optional: Set up specific services and tools within the CI/CD pipeline
  services = ["jenkins", "github_actions", "gitlab"]  # List of CI/CD services to integrate with Datadog (e.g., Jenkins, GitHub Actions, GitLab CI)

  # Optional: Specify environments that will send data to Datadog
  environments = ["production", "staging"]  # List of environments to monitor through the CI/CD pipeline

  # Optional: Configuration for monitoring build and deployment processes
  monitoring_settings {
    monitor_builds    = true  # Enable monitoring of build processes
    monitor_deployments = true  # Enable monitoring of deployment processes
  }

  # Optional: Set up alerting for failed builds or deployments
  alerting {
    enabled = true  # Enable alerting for CI/CD pipeline events
    alert_threshold = 1  # Number of failures required to trigger an alert
    notification_channel = "slack"  # Notification channel for alerts (e.g., Slack, Email)
  }

  # Optional: Tags to help categorize and filter the CI/CD integration
  tags = ["env:ci", "team:devops"]  # Tags to categorize the integration by environment and team
}
```

```
# Example of linking Datadog CI/CD integration to specific pipelines
resource "datadog_pipeline_monitor" "pipeline_monitor" {
  integration_id = datadog_integration.ci_cd.id  # Link to the CI/CD integration
  pipeline_name  = "Main Pipeline"  # Name of the CI/CD pipeline to monitor

  # Define specific metrics or conditions to monitor within the pipeline
  metrics {
    metric_name = "build_duration"  # Monitor build duration
    threshold   = 300  # Set a threshold of 300 seconds
  }

  # Optional: Set up notifications for pipeline metrics
  notification {
    type  = "slack"  # Type of notification (e.g., Slack, Email)
    channel = "#ci-cd-alerts"  # Slack channel for notifications
  }
}
```

Distributed Tracing Setup

```
# Datadog Monitor for Distributed Tracing Configuration
resource "datadog_monitor" "distributed_tracing" {
  name  = "Distributed Tracing"  # Descriptive name for the tracing monitor
  type  = "trace"              # Monitor type set to 'trace' for distributed tracing

  # Query to monitor average request duration over the last 5 minutes for a specific service
(e.g., Flask)
  query = "avg(last_5m):avg:trace.flask.request.duration{*} by {service}"

  # Thresholds for triggering alerts based on average request duration
  thresholds {
    critical = 500  # Trigger a critical alert if average request duration exceeds 500 ms
    warning  = 300  # Trigger a warning alert if average request duration exceeds 300 ms
  }

  # Custom message to be sent when the monitor is triggered
  message = <<EOF
    High average request duration detected for the Flask service.
    Please investigate the trace data to identify potential bottlenecks or errors.
    Monitor: {{monitor.name}}
    Service: {{service}}
    Duration: {{value}} ms
  EOF

  # Options for monitoring, including renotification and timeout settings
  options {
    notify_no_data           = true  # Notify if no data is available
    no_data_timeframe        = 10    # Timeframe in minutes for considering no data as an issue
    renotify_interval        = 60    # Re-notify if the issue persists after 60 minutes
    evaluation_delay         = 300   # Delay evaluation by 5 minutes to allow for data ingestion
    timeout_h                = 2     # Set a timeout of 2 hours before resolving the alert
    include_tags             = true  # Include tags in the alert notifications
    escalation_message       = "Immediate action required."  # Additional message if the alert
escalates
    require_full_window      = true  # Require the full evaluation window to trigger an alert
    new_host_delay           = 300   # Delay evaluation on newly added hosts by 5 minutes
  }

  # Notification settings, specifying channels for alert notifications
  notify {
    type     = "slack"  # Notification type (e.g., Slack)
    channel  = "#tracing-alerts"  # Slack channel for notifications
  }

  # Tagging the monitor for easier filtering and management
  tags = [
    "env:production",   # Environment tag
    "service:flask",    # Service tag
    "team:backend"      # Team responsible for this monitor
  ]

  # Optional scheduling of the monitor to limit alerts to specific timeframes
  scheduling {
    weekdays_only   = true  # Enable the monitor only on weekdays
    maintenance_windows {
      start = "03:00"  # Start maintenance window at 3 AM
      end   = "04:00"  # End maintenance window at 4 AM
    }
  }

  # Documentation or runbook URL to help the team respond to alerts effectively
  runbook_url = "https://company-runbook.example.com/monitoring/tracing"  # URL to
runbook or documentation for this monitor
}
```

Service Level Objective (SLO) Configuration

```
# Datadog Service Level Objective (SLO) Configuration
resource "datadog_slo" "slo" {
  name        = "Uptime SLO"          # Descriptive name for the SLO
  description = "Monitors service uptime to ensure 99% availability."  # Brief description of the SLO

  # Define the SLO threshold percentage for success
  threshold   = 99

  # SLO Query Definitions
  query {
    numerator   = "sum:service.uptime{*}.as_count()"        # Numerator for the SLO calculation, representing successful uptime counts
    denominator = "sum:service.uptime.total{*}.as_count()"    # Denominator for the SLO calculation, representing total uptime counts
  }

  # Optional tags for categorizing the SLO, useful for filtering and organization
  tags = [
    "env:production",     # Environment tag
    "team:operations",    # Team responsible for the SLO
    "service:core-api",   # Specific service being monitored
  ]

  # Timeframe settings for SLO evaluation
  timeframe = "30d"       # Evaluation over the last 30 days (can be set to 7d, 30d, 90d, etc.)

  # Alerting settings for when the SLO breaches the threshold
  alert {
    warning = 99.5        # Warning threshold at 99.5% uptime
    critical = 99.0       # Critical threshold at 99% uptime

    notify = {
      type = "email"      # Notification type (e.g., email)
      address = "oncall@example.com"  # Email address for notifications
    }
  }
# Optional scheduling of the SLO for specific time periods
```

```
scheduling {
    evaluation_periods = ["1d", "7d"]  # Evaluate daily and weekly
    exclude_periods = ["2024-12-25"]   # Exclude specific dates such as holidays
  }

  # Optional burn rate alerting configuration
  burn_rate_alert {
    critical = {
      threshold  = 2        # Critical burn rate threshold
      evaluation_period = "1h"  # Evaluation period for burn rate alerting
      message = "Burn rate too high for the last hour, immediate action required!"  # Custom message
    }
    warning = {
      threshold  = 1        # Warning burn rate threshold
      evaluation_period = "4h"  # Evaluation period for burn rate alerting
      message = "Burn rate elevated, please investigate."  # Custom message
    }
  }

  # Optional runbook URL for detailed resolution steps or documentation
  runbook_url = "https://company-runbook.example.com/slo-uptime"  # URL to the runbook or documentation for this SLO

  # Compliance settings (e.g., for SLOs that must meet certain regulatory standards)
  compliance {
    standard = “XXXX"  # Compliance standard
     requirements = ["99% uptime must be maintained at all times"]  # Specific requirements tied to this SLO
  }

  # Automation settings to trigger actions when SLO is breached
  automation {
    trigger = "incident.create"  # Trigger an incident creation in case of breach
    action = "page_on_call"      # Action to page the on-call team
  }
}
```

Automated Incident Management Integration

```
# Datadog PagerDuty Integration Configuration
resource "datadog_integration_pagerduty" "pagerduty" {
 # Name of the PagerDuty integration, used for identification within Datadog
 name = "PagerDuty Integration"

 # Name of the PagerDuty service to link with Datadog
 service_name = "my-service"

 # Optional Description for Clarity
 description = "Integration of PagerDuty with Datadog for automated incident management and
alerting."

 # PagerDuty Service Configuration
 pagerduty_service {
  # Required: The PagerDuty service key used for integration
  service_key = var.pagerduty_service_key

  # Optional: PagerDuty integration type (e.g., 'email' or 'api')
  integration_type = "api"

  # Optional: PagerDuty API URL if using a custom endpoint
  api_url = "https://api.pagerduty.com/"
 }

 # Notification Settings for Integration
 notifications {
  # List of Datadog monitors that will send notifications to PagerDuty
  monitor_ids = [
   "monitor_id_1",   # Example monitor ID 1
   "monitor_id_2"    # Example monitor ID 2
  ]

  # Optional: Notification level to be sent to PagerDuty (e.g., critical, warning)
  severity = "critical"

  # Optional: Custom message format for notifications
  message_format = "Incident triggered: {{ .message }}"
 }
```

```
 # Optional Tags for Categorization
 tags = [
  "env:production",      # Environment tag indicating this integration is for production
  "team:operations",     # Team responsible for operations
  "integration:pagerduty" # Integration type tag
 ]

 # Optional Automation Settings
 automation {
  # Action to take when the integration is triggered (e.g., create an incident, send alert)
  action = "incident.create"

  # Optional: Frequency for re-notifying (in minutes) if the incident is not resolved
  renotify_interval = 60
 }

 # Optional: Additional Settings
 additional_settings {
  # Optional: Specify the incident urgency (e.g., low, medium, high)
  urgency = "high"

  # Optional: Custom tags to add to PagerDuty incidents
  custom_tags = [
   "source:datadog",   # Tag indicating the source is Datadog
   "integration:example"  # Example tag for integration tracking
  ]
 }
}

# Variable definitions for PagerDuty service key (should be securely managed)
variable "pagerduty_service_key" {
 description = "The PagerDuty service key used for integration."
 type        = string
 sensitive   = true  # Mark as sensitive to avoid exposing in logs or outputs
}
```

Compliance and Governance Control

```
# Datadog Compliance Monitor Configuration
resource "datadog_compliance_monitor" "compliance" {
  name       = "Compliance Check"            # Descriptive name for the compliance monitor
  type       = "audit"                       # Type of monitor set to 'audit'

  # Query Definition
  query      = "avg(last_5m):avg:service.uptime{*} > 99"  # Query to monitor service uptime to
ensure it meets compliance standards

  # Optional Tags for Categorization
  tags       = [
    "env:production",      # Environment tag indicating this monitor is for production
    "team:compliance",     # Team responsible for compliance
    "service:core-api",    # Specific service being monitored for compliance
  ]

  # Optional Description for Clarity
  description = "This monitor checks if the service uptime is consistently above 99% over the
last 5 minutes to ensure compliance with service level agreements."

  # Notification Settings
  notification {
    notify      = {
      type    = "email"                # Notification type (e.g., email, webhook)
      address = "compliance-team@example.com"  # Email address for notifications
    }
    severity    = "critical"             # Severity level of the notification
    message     = "Service uptime is below the compliance threshold of 99%."  # Custom
message to be sent in notifications
  }
```

```
  # Optional Thresholds for Alerting
  thresholds {
    warning  = 98.5                  # Warning threshold for uptime
    critical = 99.0                  # Critical threshold for uptime
  }
# Scheduling and Evaluation Periods
  scheduling {
    evaluation_periods = ["1h", "24h"]    # Evaluate compliance over the last hour and day
    exclude_periods    = ["2024-12-25"]   # Exclude specific dates like holidays from compliance
checks
  }

  # Runbook URL for Incident Resolution
  runbook_url = "https://company-runbook.example.com/compliance-check"  # URL to the
runbook or documentation for compliance checks

  # Optional Automation Settings
  automation {
    trigger = "incident.create"       # Trigger an incident creation if the compliance check fails
    action  = "page_on_call"          # Action to page the on-call team for immediate attention
  }

  # Compliance Standard Information
  compliance {
    standard = "ISO-27001"           # Compliance standard this monitor is checking against
    requirements = [
      "Uptime must be above 99% at all times to meet compliance requirements."
    ]  # Specific compliance requirements associated with the SLO
  }
}
```

Maintenance Window Setup (Suppress Alerts During Scheduled Outages)

```
# Datadog Downtime Configuration for Scheduled Maintenance
resource "datadog_downtime" "maintenance_window" {
  # Scope of the downtime, applying to all monitors in Datadog
  scope = ["*"]

  # Start time of the downtime in Unix timestamp format
  start = 1657296000  # Example: July 10, 2022 00:00:00 GMT

  # End time of the downtime in Unix timestamp format
  end = 1657303200    # Example: July 10, 2022 02:00:00 GMT

  # Message to display during the downtime
  message = "Scheduled maintenance window. All monitors will be temporarily disabled."

  # Recurrence configuration for recurring downtimes
  recurrence {
    # Type of recurrence, e.g., 'weeks', 'days', 'months'
    type = "weeks"

    # Period of recurrence, e.g., every 1 week
    period = 1

    # Days of the week on which the downtime occurs, e.g., Monday
    week_days = ["Monday"]

    # Optional: Specify the time of day when the downtime should start and end
    start_time = "00:00"  # Start time in HH:MM format (optional)
    end_time = "02:00"    # End time in HH:MM format (optional)

    # Optional: Define specific time zones if required
    timezone = "UTC"      # Time zone for the downtime period (optional)
  }

  # Optional: List of tags to filter which monitors are affected by the downtime
  tags = [
    "env:production",    # Example tag indicating the environment
    "team:operations"    # Example tag indicating the responsible team
  ]

  # Optional: Notify specific teams or channels about the downtime
  notifications {
    email = ["admin@example.com"]  # List of email addresses to notify
    slack = ["#operations"]        # List of Slack channels to notify
  }

  # Optional: Add a URL for more information about the downtime
  more_info_url = "https://example.com/maintenance"  # URL with additional details
}

# Variable Definitions
variable "start_time_unix" {
  description = "Unix timestamp for the start of downtime."
  type        = number
  default     = 1657296000  # Example timestamp
}

variable "end_time_unix" {
  description = "Unix timestamp for the end of downtime."
  type        = number
  default     = 1657303200  # Example timestamp
}
```

Measure Progress with Business Outcomes

- **Mean Time to Detect (MTTD):** Decrease the time it takes to identify issues.
- **Mean Time to Resolve (MTTR)**: Shorten the time it takes to detect and fix issues.
- Mean Time Between Failures (MTBF): Increase the interval between system failures.
- Improved System Reliability and Availability: Enhance system uptime and minimize downtime.
- Enhanced User Experience: Boost user satisfaction with faster and smoother interactions.
- Optimized Resource Utilization: Ensure efficient use of computing resources to save costs.
- Increased Development Velocity: Accelerate the delivery of new features and updates.
- Alignment with Service Level Objectives (SLOs): Ensure observability efforts meet defined performance targets and business objectives.

Best practices

- **Secure API Key Management:** Use secret management tools (e.g., AWS Secrets Manager, HashiCorp Vault) to securely manage and rotate Datadog API keys and other sensitive credentials, preventing unauthorized access and ensuring compliance with security best practices.
- **Use Version Control:** Store all observability configurations in a version control system like Git for traceability and easy rollback of changes.
- **Modularize Configurations:** Break down configurations into reusable modules (e.g., monitors, dashboards, log pipelines) to promote consistency and adaptability across environments.
- **Automate Deployments:** Integrate observability code into your CI/CD pipeline to automate the deployment and ensure configurations are consistent and up-to-date.
- **Implement Testing and Validation:** Include automated tests to validate the correctness of observability configurations before deploying them to production.
- **Enable Role-Based Access Control (RBAC)**: Implement access controls to restrict and manage who can modify and deploy observability configurations, ensuring security and governance.
- **Centralize Logging and Tracing:** Ensure all logs, traces, and metrics are centralized in Datadog for easy correlation, aiding in quicker troubleshooting and root cause analysis.
- **Standardize Alerting and Monitoring Practices:** Define standard alerting thresholds, SLOs, and escalation paths to avoid alert fatigue and ensure that critical incidents are prioritized.
- **Document and Review Configurations Regularly:** Maintain comprehensive documentation and periodically review observability configurations to keep them aligned with infrastructure and application changes.
- **Leverage Datadog Integrations:** Fully utilize Datadog's integrations (e.g., with AWS, Kubernetes) to enhance observability data and automate monitoring tasks.
- **Foster Collaboration Across Teams:** Involve DevOps, SRE, and development teams in the observability-as-code process to ensure configurations meet the needs of all stakeholders and are aligned with organizational goals.

Thank you.