Problem

We face a growing challenge as organizations adopt multicloud strategies. Monitoring the health and performance of resources across different cloud platforms becomes increasingly complex. Each cloud provider offers unique monitoring tools and APIs, making it difficult to consolidate and centralize monitoring data. Manually configuring and managing monitoring solutions for each platform is time-consuming and prone to errors.

Solution

To address this, we leverage Terraform to automate the deployment and configuration of a multicloud monitoring solution. Using Terraform's provider ecosystem and cross-platform capabilities, we create a centralized solution that aggregates monitoring data from multiple cloud providers into a unified platform such as Datadog. Below is an example of how we implement this solution using Terraform, AWS, Azure, and Datadog.

Implementation

1. Providers Configuration

We define providers for AWS, Azure, and Datadog to interact with their respective APIs. This ensures that Terraform can manage resources across multiple platforms.

```
provider "aws" {
  region = "us-west-2"
}

provider "azurerm" {
  features {}
}

provider "datadog" {
  api_key = var.datadog_api_key
  app_key = var.datadog_app_key
}
```

---

2. Deploying AWS Resources

We deploy an EC2 instance on AWS as an example resource to monitor.

```
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t3.micro"

  tags = {
    Name = "example-aws-instance"
  }
}
```

---

3. Deploying Azure Resources

We provision an Azure Resource Group and Virtual Machine. The VM includes basic network configurations and is set up for monitoring.

```
resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "East US"
}

resource "azurerm_virtual_machine" "example" {
  name                  = "example-vm"
  location              = azurerm_resource_group.example.location
  resource_group_name   = azurerm_resource_group.example.name
  network_interface_ids = [azurerm_network_interface.example.id]
  vm_size               = "Standard_DS1_v2"
}
```

---

4. Configuring Datadog Monitors

We create monitors in Datadog to track CPU usage for both the AWS and Azure resources. These monitors trigger alerts when CPU usage exceeds 80%.

AWS Monitor:

```
resource "datadog_monitor" "aws_cpu" {
  name    = "High CPU Usage - AWS"
  type    = "metric alert"
  message = "CPU usage is high on AWS instance {{host.name}}."
  query   = "avg(last_5m):avg:aws.ec2.cpu{host:${aws_instance.example.id}} >
80"

  monitor_thresholds {
    critical = 80
  }
}
```

Azure Monitor:

```
resource "datadog_monitor" "azure_cpu" {
  name    = "High CPU Usage - Azure"
  type    = "metric alert"
  message = "CPU usage is high on Azure VM {{host.name}}."
  query   =
"avg(last_5m):avg:azure.vm.percentage_cpu{resource_group:${azurerm_resource_gr
oup.example.name},name:${azurerm_virtual_machine.example.name}} > 80"

  monitor_thresholds {
    critical = 80
  }
}
```

---

5. Creating a Datadog Dashboard

We create a unified dashboard in Datadog to visualize CPU usage across AWS and Azure. This provides a centralized view of multicloud performance.

```
resource "datadog_dashboard" "multi_cloud" {
  title       = "Multi-Cloud Overview"
  description = "Overview of AWS and Azure resources"
  layout_type = "ordered"

  widget {
    timeseries_definition {
      title = "CPU Usage - AWS vs Azure"
      request {
```

```
        q            = "avg:aws.ec2.cpu{host:${aws_instance.example.id}}"
        display_type = "line"
      }
      request {
        q            =
"avg:azure.vm.percentage_cpu{resource_group:${azurerm_resource_group.example.n
ame},name:${azurerm_virtual_machine.example.name}}"
        display_type = "line"
      }
    }
  }
}

output "datadog_dashboard_url" {
  value =
"https://app.datadoghq.com/dashboard/${datadog_dashboard.multi_cloud.id}"
}
```

Discussion

Deploying a multicloud monitoring solution using Terraform offers several benefits:

1. **Centralized Monitoring**
   By using Datadog, we aggregate data from AWS and Azure into a unified interface, simplifying resource management.
2. **Consistency**
   Terraform ensures monitoring configurations are consistent across platforms, reducing manual errors.
3. **Automation**
   Automating the deployment of monitoring solutions saves time and minimizes configuration errors.
4. **Version Control**
   Monitoring configurations are version-controlled, enabling us to track changes and roll back if needed.
5. **Scalability**
   As our infrastructure grows, Terraform makes it easy to add resources and update monitoring configurations.

Best Practices

- **Use Variables**: Use Terraform variables to make configurations flexible across environments.
- **Modularize**: Organize code into reusable modules for each cloud provider and monitoring setup.
- **Tag Resources**: Apply consistent tagging to organize and filter resources effectively.
- **Secure Credentials**: Use secure methods like environment variables or secrets management for API keys.
- **Monitor Costs**: Track expenses for multicloud deployments and Datadog usage.
- **Stay Updated**: Regularly update Terraform configurations and provider versions for the latest features.
- **Implement Alerts**: Configure alerts to proactively notify teams of potential issues.
- **Ensure Compliance**: Align monitoring with organizational or industry compliance standards.

Summary

By automating multicloud monitoring with Terraform, we streamline resource management and gain a unified view of infrastructure across platforms. This approach enhances operational efficiency, reduces errors, and scales seamlessly with growing environments.