You're looking for a robust, production-ready, and reusable Terraform codebase for your Azure AI Hub accelerator, incorporating Azure OpenAI and all the best practices we've discussed, including automation, cost management, and security.
This will be a comprehensive setup, so I'll structure it clearly with a focus on modularity and environment segregation.

## Project Structure (Standard Terraform Best Practice)

```
.
├── environments/
│   ├── dev/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── backend.tf
│   ├── prod/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── backend.tf
│   └── outputs.tf  # Optional: For cross-environment references if
any, or general info
├── modules/
│   ├── networking/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── shared_services/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── monitoring/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── aihub/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── apim/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   └── policies/ # Directory for APIM policies (e.g., XML files)
│   │       └── openai_rewrite_policy.xml
│   ├── cost_management/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── function_app_cost_processor/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   │
│   │       └── src/ # Source code for the Function App
│   │           └── __init__.py
│   │           └── function_app.py
│   │           └── requirements.txt
│   └── dns/ # If you manage private DNS zones here
│       ├── main.tf
│       ├── variables.tf
│       └── outputs.tf
└── main.tf  # Optional, usually in a root for global
settings/providers
```

## Core Concepts & Production Best Practices Implemented:

1. **Modularity:** Each distinct Azure service or logical grouping is a separate module (networking, shared_services, aihub, apim, etc.).
2. **Environment Segregation:** Dedicated environments folders (dev, prod) to manage environment-specific configurations (.tfvars indirectly via variables.tf and passed values).
3. **Remote State Backend:** Using Azure Storage Accounts for secure, shared, and versioned Terraform state (critical for teams and CI/CD).
4. **Security:**
    - **Private Endpoints:** Implemented for Azure AI Services, APIM, Key Vault, Storage Account, and Function App (where applicable). This ensures traffic stays within your VNet.
    - **Managed Identities:** Used for APIM to access Key Vault and for the Function App to access Event Hubs and Storage, minimizing credential exposure.
    - **Key Vault Integration:** Centralized secret management for API keys (e.g., OpenAI API key if not using managed identity for backend calls) and other sensitive values.
    - **Network Security Groups (NSGs):** Implicitly managed by subnets and private endpoints, but could be explicitly defined for granular control.
    - **Access Policies (RBAC):** Terraform implicitly creates necessary permissions, but explicit azurerm_role_assignment is used for clarity and critical cross-service access.
5. **Reusability:** Modules are designed to be generic and reusable across environments and even subscriptions (with proper provider configuration).
6. **Cost Management:** Placeholder for Event Hubs to stream billing data, consumed by an Azure Function App, for Power BI reporting.
7. **Automation Readiness (CI/CD):** The structure is ideal for Azure DevOps Pipelines or GitHub Actions.
    - Separate pipelines per environment.
    - terraform plan and terraform apply stages.
    - Variable groups for environment-specific secrets/values in the pipeline.
    - Service Principals with least privilege for Terraform execution.

## Detailed Code (Selected Key Modules & Environment)

**Assumptions:**

- **Existing VNet:** You have an existing VNet and subnets. We will use data blocks to reference them.
- **Naming Convention:** A consistent naming convention (e.g., resourcetype-env-purpose) is used.

- **Secrets:** Sensitive values like the OpenAI API key are managed in Azure Key Vault.

## 1. main.tf (Root - For Provider Configuration)

```
# main.tf
terraform {
  required_version = ">= 1.5.0" # Ensure compatibility with newer
Azure resources

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0" # Use a compatible version, check Terraform
Registry
    }
  }
}

provider "azurerm" {
  features {} # Required for current azurerm provider versions
  # Authenticate using Azure CLI, Service Principal, or Managed
Identity
  # For CI/CD, a Service Principal is common:
  # client_id       = var.arm_client_id
  # client_secret   = var.arm_client_secret
  # tenant_id       = var.arm_tenant_id
  # subscription_id = var.arm_subscription_id
}

# --- Root Variables (if any, typically for global settings) ---
variable "arm_subscription_id" {
  description = "The Azure Subscription ID to deploy resources into."
  type        = string
  sensitive   = true
}

# Add other ARM authentication variables if using Service Principal
directly here
# variable "arm_client_id" {}
# variable "arm_client_secret" {}
# variable "arm_tenant_id" {}
```

## 2. environments/dev/backend.tf (Remote State Configuration)

Create a storage account and container manually *once* for each environment to store the Terraform state.

```
# environments/dev/backend.tf
terraform {
  backend "azurerm" {
    resource_group_name  = "rg-tfstate-dev" # Manually created once
    storage_account_name = "tfstatedevaihub" # Manually created once,
globally unique
    container_name       = "tfstate"
    key                  = "aihub-dev.terraform.tfstate"
    # Ensure the Service Principal running Terraform has Storage Blob
Data Contributor on this storage account
  }
}
```

## 3. environments/dev/variables.tf (Environment-Specific Variables)

```
# environments/dev/variables.tf

variable "location" {
  description = "The Azure region for the Dev environment."
  type        = string
  default     = "uksouth"
}

variable "environment" {
  description = "The deployment environment name."
  type        = string
  default     = "dev"
}

variable "resource_group_prefix" {
  description = "Prefix for resource group names."
  type        = string
  default     = "rg-dev-aihub"
}

# Networking
variable "existing_vnet_name" {
  description = "Name of the existing VNet."
  type        = string
  default     = "vnet-shared-dev"
}

variable "existing_vnet_resource_group_name" {
  description = "Resource group name of the existing VNet."
  type        = string
  default     = "rg-network-dev"
}

variable "existing_subnets" {
  description = "Map of existing subnet names to their purpose."
  type = map(string)
  default = {
    "apim_subnet" = "snet-apim-dev"
    "aihub_subnet" = "snet-aihub-dev" # For AI Services Private
Endpoint
    "functions_subnet" = "snet-functions-dev" # For Function App
Private Endpoint
    "storage_subnet" = "snet-storage-dev" # For Storage Account
Private Endpoint
    "keyvault_subnet" = "snet-keyvault-dev" # For Key Vault Private
Endpoint
  }
}

# AI Hub (Azure AI Services)
variable "ai_services_sku" {
  description = "The SKU for the Azure AI Services account (e.g.,
'S0', 'F0'). Use F0 for dev if acceptable."
  type        = string
  default     = "S0" # S0 is production-ready, F0 is free tier (limits
apply)
}

variable "openai_model_deployments_dev" {
  description = "A map of Azure OpenAI model deployments to create in
Dev. Key is deployment name, value is model config."
  type = map(object({
    model_name    = string
    model_version = string
    capacity      = number
  }))
  default = {
    "gpt-35-turbo-dev" = {
      model_name    = "gpt-35-turbo"
      model_version = "1106" # Specific version
      capacity      = 1
    }
    # "gpt-4o-dev" = { # Add GPT-4o if needed, ensure quota available
    #   model_name    = "gpt-4o"
    #   model_version = "2024-05-13"
    #   capacity      = 1
    # }
  }
}

variable "openai_api_version" {
  description = "The API version to use for Azure OpenAI calls (e.g.,
'2024-02-01')."
  type        = string
  default     = "2024-02-01" # Standard API version for Chat
Completions
}


# APIM
variable "apim_sku_name" {
  description = "SKU for API Management (e.g., 'Developer_1',
'Premium_1'). 'Developer' for dev/test."
  type        = string
  default     = "Developer_1"
}

variable "apim_products" {
  description = "APIM products to create."
  type = map(object({
    display_name = string
    description  = string
    state        = string
    subscription_required = bool
    approval_required     = bool
  }))
  default = {
    "open-ai-models" = {
      display_name = "OpenAI Models"
      description  = "Access to various Azure OpenAI models."
      state        = "published"
      subscription_required = true
      approval_required     = true
    }
  }
}

# Cost Management
variable "cost_management_event_hub_namespace_name" {
  description = "Name for the Event Hub Namespace for cost data."
  type        = string
  default     = "evhns-cost-dev"
}

variable "cost_management_event_hub_name" {
  description = "Name for the Event Hub for cost data."
  type        = string
  default     = "costdata"
}

variable "cost_management_consumption_api_scope" {
  description = "The scope for the Azure Consumption API export (e.g.,
'/subscriptions/YOUR_SUB_ID')."
  type        = string
  default     = "/subscriptions/00000000-0000-0000-0000-000000000000"
# REPLACE WITH YOUR SUBSCRIPTION ID
}

# Function App for Cost Processing
variable "function_app_name" {
  description = "Name of the Function App for cost processing."
  type        = string
  default     = "func-costprocessor-dev"
```

```
}

variable "function_app_storage_sku" {
  description = "SKU for the Function App storage account."
  type        = string
  default     = "Standard_LRS"
}

variable "function_app_python_version" {
  description = "Python version for the Function App."
  type        = string
  default     = "3.9"
}

# Shared Services (Key Vault, Storage)
variable "key_vault_sku_name" {
  description = "SKU for Key Vault (Standard or Premium)."
  type        = string
  default     = "Standard"
}
```

## 4. environments/dev/main.tf (Orchestration for Dev Environment)

This file stitches together all the modules.

```
# environments/dev/main.tf

# Providers and Backend already configured at root/env level

# --- Data Sources for Existing Resources ---

# Existing Virtual Network
data "azurerm_virtual_network" "main_vnet" {
  name                = var.existing_vnet_name
  resource_group_name = var.existing_vnet_resource_group_name
}

# Existing Subnets
data "azurerm_subnet" "apim_subnet" {
  name                 = var.existing_subnets["apim_subnet"]
  virtual_network_name = data.azurerm_virtual_network.main_vnet.name
  resource_group_name  =
data.azurerm_virtual_network.main_vnet.resource_group_name
}

data "azurerm_subnet" "aihub_subnet" {
  name                 = var.existing_subnets["aihub_subnet"]
  virtual_network_name = data.azurerm_virtual_network.main_vnet.name
  resource_group_name  =
data.azurerm_virtual_network.main_vnet.resource_group_name
}
```

```
data "azurerm_subnet" "functions_subnet" {
  name                 = var.existing_subnets["functions_subnet"]
  virtual_network_name = data.azurerm_virtual_network.main_vnet.name
  resource_group_name  =
data.azurerm_virtual_network.main_vnet.resource_group_name
}

data "azurerm_subnet" "storage_subnet" {
  name                 = var.existing_subnets["storage_subnet"]
  virtual_network_name = data.azurerm_virtual_network.main_vnet.name
  resource_group_name  =
data.azurerm_virtual_network.main_vnet.resource_group_name
}

data "azurerm_subnet" "keyvault_subnet" {
  name                 = var.existing_subnets["keyvault_subnet"]
  virtual_network_name = data.azurerm_virtual_network.main_vnet.name
  resource_group_name  =
data.azurerm_virtual_network.main_vnet.resource_group_name
}


# --- Resource Group for this Deployment ---
resource "azurerm_resource_group" "main_rg" {
  name     = "${var.resource_group_prefix}-main"
  location = var.location
  tags = {
    environment = var.environment
    project     = "AIHubAccelerator"
  }
}

# --- Module Calls ---

# 1. Shared Services (Key Vault, common Storage for functions/logs)
module "shared_services" {
  source = "../../modules/shared_services"

  resource_group_name = azurerm_resource_group.main_rg.name
  location            = var.location
  environment         = var.environment

  key_vault_sku_name  = var.key_vault_sku_name
  key_vault_subnet_id = data.azurerm_subnet.keyvault_subnet.id
  storage_subnet_id   = data.azurerm_subnet.storage_subnet.id

  # Provide initial secret for OpenAI API Key (This would be set by a
pipeline/secret management system)
  # For production, this secret should ideally be inserted by the
pipeline from a secure source
  openai_api_key      = "YOUR_OPENAI_API_KEY_SECRET_VALUE" #
IMPORTANT: REPLACE IN PROD!!!
}

# 2. Monitoring (Log Analytics, Application Insights)
module "monitoring" {
  source = "../../modules/monitoring"

  resource_group_name = azurerm_resource_group.main_rg.name
  location            = var.location
  environment         = var.environment
}

# 3. AI Hub (Azure AI Services for OpenAI)
module "aihub" {
  source = "../../modules/aihub"

  resource_group_name        = azurerm_resource_group.main_rg.name
  location                   = var.location
  environment                = var.environment
  ai_services_sku            = var.ai_services_sku
  openai_model_deployments   = var.openai_model_deployments_dev # Use
dev specific deployments
  log_analytics_workspace_id =
module.monitoring.log_analytics_workspace_id
  aihub_subnet_id            = data.azurerm_subnet.aihub_subnet.id
}

# 4. API Management
module "apim" {
  source = "../../modules/apim"

  resource_group_name        = azurerm_resource_group.main_rg.name
  location                   = var.location
  environment                = var.environment
  apim_subnet_id             = data.azurerm_subnet.apim_subnet.id
  apim_sku_name              = var.apim_sku_name

  # Inputs from other modules
  ai_services_endpoint         = module.aihub.ai_services_endpoint
  openai_api_key_secret_id     =
module.shared_services.openai_api_key_secret_id # Key Vault Secret ID
  log_analytics_workspace_id   =
module.monitoring.log_analytics_workspace_id
  application_insights_id      =
module.monitoring.application_insights_id

  apim_products                        = var.apim_products
  openai_model_deployments_to_expose =
var.openai_model_deployments_dev # Expose all deployed OpenAI models
  openai_api_version                   = var.openai_api_version
}
```

```
# 5. Cost Management (Event Hubs)
module "cost_management" {
  source = "../../modules/cost_management"

  resource_group_name        = azurerm_resource_group.main_rg.name
  location                   = var.location
  environment                = var.environment
  event_hub_namespace_name   =
var.cost_management_event_hub_namespace_name
  event_hub_name             = var.cost_management_event_hub_name
  consumption_api_scope      =
var.cost_management_consumption_api_scope
  log_analytics_workspace_id =
module.monitoring.log_analytics_workspace_id
}

# 6. Function App for Cost Processing
module "function_app_cost_processor" {
  source = "../../modules/function_app_cost_processor"

  resource_group_name        = azurerm_resource_group.main_rg.name
  location                   = var.location
  environment                = var.environment
  function_app_name          = var.function_app_name
  function_app_storage_sku   = var.function_app_storage_sku
  function_app_python_version = var.function_app_python_version
  functions_subnet_id        = data.azurerm_subnet.functions_subnet.id
# For VNet integration/Private Endpoint

  event_hub_namespace_id     =
module.cost_management.event_hub_namespace_id
  event_hub_name             = var.cost_management_event_hub_name
  log_analytics_workspace_id =
module.monitoring.log_analytics_workspace_id

  # Private endpoint for the storage account created by the function
app module
  # If the function app module itself creates a storage account, its
private endpoint might need to be configured there.
  # Otherwise, pass the storage subnet for its integration.
}

# --- Outputs (for easy reference from outside) ---

output "ai_services_endpoint" {
  description = "The endpoint for the Azure AI Services Account in
${var.environment}."
  value       = module.aihub.ai_services_endpoint
}

output "apim_gateway_url" {
  description = "The URL of the API Management Gateway in
${var.environment}."
  value       = module.apim.apim_gateway_url
}

output "key_vault_uri" {
  description = "The URI of the Key Vault in ${var.environment}."
  value       = module.shared_services.key_vault_uri
}

output "log_analytics_workspace_id" {
  description = "The ID of the Log Analytics Workspace in
${var.environment}."
  value       = module.monitoring.log_analytics_workspace_id
}

output "cost_management_event_hub_connection_string" {
  description = "The connection string for the Cost Management Event
Hub (for Power BI integration)."
  value       =
module.cost_management.event_hub_auth_rule_primary_connection_string
  sensitive   = true
}
```

## 5. modules/networking/main.tf (Optional: If you create VNets/Subnets)

If you *do* want to create your VNets and subnets with Terraform rather than referencing existing ones, this module would contain:

```
# modules/networking/main.tf (Example - if you create networking)

resource "azurerm_virtual_network" "main" {
  name                = "vnet-${var.environment}-
${var.vnet_name_suffix}"
  address_space       = var.vnet_address_space
  location            = var.location
  resource_group_name = var.resource_group_name
}

resource "azurerm_subnet" "apim" {
  name                 = "snet-apim-${var.environment}"
  resource_group_name  = var.resource_group_name
  virtual_network_name = azurerm_virtual_network.main.name
  address_prefixes     = var.apim_subnet_address_prefix
  # delegation { # If using Azure Functions Premium with VNet
integration
  #   name = "Microsoft.Web/serverFarms"
  #   service_delegation {
  #     name = "Microsoft.Web/serverFarms"
  #     actions =
["Microsoft.Network/virtualNetworks/subnets/join/action"]
  #   }
  # }
}

# ... create other subnets similarly
```

## 6. modules/shared_services/main.tf

```
# modules/shared_services/main.tf

resource "azurerm_key_vault" "main" {
  name                        = "kv-${var.environment}-aihub"
  location                    = var.location
  resource_group_name         = var.resource_group_name
  tenant_id                   =
data.azurerm_client_config.current.tenant_id
  sku_name                    = var.key_vault_sku_name
  enabled_for_disk_encryption = false # Adjust as needed
  purge_protection_enabled    = true # Recommended for production
  soft_delete_retention_days  = 7   # Adjust as per policy

  network_acl {
    default_action = "Deny" # Lock down access
    virtual_network_subnet_ids = [var.key_vault_subnet_id] # Allow
from specific subnet
    bypass = "AzureServices" # Allow trusted Azure services
  }

  tags = {
    environment = var.environment
    project     = "AIHub"
    service     = "KeyVault"
  }
}

# Store OpenAI API Key as a secret
# In production, this should ideally be populated by an external
secrets management tool/pipeline
resource "azurerm_key_vault_secret" "openai_api_key" {
  name         = "openai-api-key"
  value        = var.openai_api_key # This value MUST come from a
secure source in CI/CD
  key_vault_id = azurerm_key_vault.main.id

  content_type = "text/plain" # Or appropriate content type
}

# Add a storage account for general purposes (e.g., Function App code,
temporary data)
resource "azurerm_storage_account" "general_purpose" {
  name                     = "st${var.environment}aihub" # Max 24
```

```
chars, lowercase
  resource_group_name      = var.resource_group_name
  location                 = var.location
  account_tier             = "Standard"
  account_replication_type = "LRS" # Or GRS/ZRS for higher
availability

  network_rules {
    default_action = "Deny"
    ip_rules       = [] # No direct public IP access
    virtual_network_subnet_ids = [
      var.storage_subnet_id # Allow from dedicated storage subnet
    ]
    bypass = ["AzureServices"] # Allow trusted Azure services
  }

  tags = {
    environment = var.environment
    project     = "AIHub"
    service     = "Storage"
  }
}

# Data source for the current Azure client configuration (for
tenant_id)
data "azurerm_client_config" "current" {}

output "key_vault_id" {
  value = azurerm_key_vault.main.id
}

output "key_vault_uri" {
  value = azurerm_key_vault.main.vault_uri
}

output "openai_api_key_secret_id" {
  value     = azurerm_key_vault_secret.openai_api_key.id
  sensitive = true
}

output "general_purpose_storage_account_name" {
  value = azurerm_storage_account.general_purpose.name
}

output "general_purpose_storage_account_id" {
  value = azurerm_storage_account.general_purpose.id
}

# modules/shared_services/variables.tf

variable "resource_group_name" {
  description = "The name of the resource group."
  type        = string
}

variable "location" {
  description = "The Azure region."
  type        = string
}

variable "environment" {
  description = "The deployment environment."
  type        = string
}

variable "key_vault_sku_name" {
  description = "SKU for Key Vault (Standard or Premium)."
  type        = string
}

variable "key_vault_subnet_id" {
  description = "The ID of the subnet for Key Vault's private
endpoint."
  type        = string
}

variable "storage_subnet_id" {
  description = "The ID of the subnet for Storage Account's private
endpoint."
  type        = string
}

variable "openai_api_key" {
  description = "The secret value for the OpenAI API Key. Should be
populated from a secure source in production."
  type        = string
  sensitive   = true
}
```

## 7. modules/monitoring/main.tf

```
# modules/monitoring/main.tf

resource "azurerm_log_analytics_workspace" "main" {
  name                = "log-${var.environment}-aihub"
  location            = var.location
  resource_group_name = var.resource_group_name
  sku                 = "PerGB2018" # Or "Consumption" for smaller
scale, "CapacityReservation" for large scale
  retention_in_days   = 30 # Adjust based on compliance needs

  tags = {
    environment = var.environment
    project     = "AIHub"
    service     = "LogAnalytics"
  }
}

resource "azurerm_application_insights" "main" {
  name                = "appi-${var.environment}-aihub"
  location            = var.location
  resource_group_name = var.resource_group_name
  application_type    = "web" # Generic type, adjust if more specific
  workspace_id        = azurerm_log_analytics_workspace.main.id

  tags = {
    environment = var.environment
    project     = "AIHub"
    service     = "AppInsights"
  }
}

output "log_analytics_workspace_id" {
  value = azurerm_log_analytics_workspace.main.id
}

output "application_insights_id" {
  value = azurerm_application_insights.main.id
}

output "application_insights_instrumentation_key" {
  value     = azurerm_application_insights.main.instrumentation_key
  sensitive = true
}

# modules/monitoring/variables.tf

variable "resource_group_name" {
  description = "The name of the resource group."
  type        = string
}

variable "location" {
  description = "The Azure region."
  type        = string
}

variable "environment" {
  description = "The deployment environment."
  type        = string
}
```

## 8. modules/aihub/main.tf (Azure AI Services for OpenAI)

```
# modules/aihub/main.tf

resource "azurerm_ai_services" "main" {
  name                = "aisvc-${var.environment}-aihub"
  location            = var.location
  resource_group_name = var.resource_group_name
  sku_name            = var.ai_services_sku

  # Network ACLs for Private Endpoint integration
  network_acls {
    default_action             = "Deny"
    virtual_network_subnet_ids = [var.aihub_subnet_id]
    bypass                     = "AzureServices"
  }

  tags = {
    environment = var.environment
    project     = "AIHub"
    service     = "AIServices"
  }
}

# Private Endpoint for Azure AI Services
resource "azurerm_private_endpoint" "ai_services_pe" {
  name                = "pe-${azurerm_ai_services.main.name}"
  location            = var.location
  resource_group_name = var.resource_group_name
  subnet_id           = var.aihub_subnet_id

  private_service_connection {
    name                           = "psc-
${azurerm_ai_services.main.name}"
    is_manual_connection           = false
    private_connection_resource_id = azurerm_ai_services.main.id
    subresource_names              = ["account"] # For Azure AI
Services
  }

  tags = {
    environment = var.environment
    project     = "AIHub"
  }
}

# Private DNS Zone and Link for Azure AI Services
# The specific zone name for Azure AI Services is often
"privatelink.cognitiveservices.azure.com"
resource "azurerm_private_dns_zone" "ai_services_dns_zone" {
  name                = "privatelink.cognitiveservices.azure.com"
  resource_group_name = var.resource_group_name # Or a dedicated DNS
RG
}

resource "azurerm_private_dns_zone_virtual_network_link"
"ai_services_dns_link" {
  name                  = "link-${azurerm_ai_services.main.name}"
  resource_group_name   = var.resource_group_name
  private_dns_zone_name =
azurerm_private_dns_zone.ai_services_dns_zone.name
  virtual_network_id    = var.vnet_id
  registration_enabled  = false # No direct VM registration needed
here
}

resource "azurerm_private_dns_a_record" "ai_services_a_record" {
  name                = azurerm_ai_services.main.name
  zone_name           =
azurerm_private_dns_zone.ai_services_dns_zone.name
  resource_group_name = var.resource_group_name
  ttl                 = 300
  records             =
azurerm_private_endpoint.ai_services_pe.private_service_connection[0].
private_ip_address
}


# Deploy Azure OpenAI models within the Azure AI Services Account
resource "azurerm_cognitive_deployment" "openai_deployments" {
  for_each = var.openai_model_deployments

  name                 = each.key
  cognitive_account_id = azurerm_ai_services.main.id
  model {
    format  = "OpenAI"
    name    = each.value.model_name
    version = each.value.model_version
  }
  scale {
    type     = "Standard"
    capacity = each.value.capacity
  }

  tags = {
    environment = var.environment
    deployment  = each.key
  }
}

# Diagnostics for Azure AI Services
resource "azurerm_monitor_diagnostic_setting"
"ai_services_diagnostics" {
  name                       = "ai-services-diagnostics"
  target_resource_id         = azurerm_ai_services.main.id
  log_analytics_workspace_id = var.log_analytics_workspace_id

  log {
    category = "Audit"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30 # Match LA workspace retention
    }
  }
  log {
    category = "Requests"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30
    }
  }
  metric {
    category = "AllMetrics"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30
    }
  }
}

output "ai_services_account_id" {
  value       = azurerm_ai_services.main.id
  description = "The ID of the Azure AI Services Account."
}

output "ai_services_endpoint" {
  value       = azurerm_ai_services.main.endpoint
  description = "The endpoint URL for the Azure AI Services Account."
}

# modules/aihub/variables.tf

variable "resource_group_name" {
  description = "The name of the resource group for AI Hub."
  type        = string
}

variable "location" {
  description = "The Azure region for AI Hub."
  type        = string
}
```

```
variable "environment" {
  description = "The deployment environment (e.g., dev, qa, prod)."
  type        = string
}

variable "ai_services_sku" {
  description = "The SKU for the Azure AI Services account (e.g.,
'S0', 'F0')."
  type        = string
}

variable "openai_model_deployments" {
  description = "A map of Azure OpenAI model deployments to create.
Key is deployment name, value is model config."
  type = map(object({
    model_name    = string
    model_version = string
    capacity      = number
  }))
}

variable "log_analytics_workspace_id" {
  description = "The ID of the Log Analytics Workspace for AI Services
diagnostics."
  type        = string
}

variable "aihub_subnet_id" {
  description = "The ID of the subnet for the Azure AI Services
Private Endpoint."
  type        = string
}

variable "vnet_id" {
  description = "The ID of the Virtual Network where the AI Hub subnet
resides."
  type        = string # Needed for Private DNS Zone VNet link
}
```

## 9. modules/apim/main.tf

```
# modules/apim/main.tf

resource "azurerm_api_management_service" "apim" {
  name                = "apim-${var.environment}-aihub"
  location            = var.location
  resource_group_name = var.resource_group_name
  publisher_name      = "Your Company"
  publisher_email     = "apim-admin@yourcompany.com"
  sku_name            = var.apim_sku_name
  # Enable Managed Identity for Key Vault access and potentially
backend calls
  identity {
    type = "SystemAssigned"
  }

  # VNet Integration for Private Endpoint to Backends (Azure AI
Services)
  # This is the VNet where APIM's internal components will reside.
  # It must be a dedicated subnet.
  virtual_network_configuration {
    subnet_id = var.apim_subnet_id
  }
  virtual_network_type = "External" # Or "Internal" if only internal
access needed

  tags = {
    environment = var.environment
    project     = "AIHub"
    service     = "APIM"
  }
}

# Role Assignment for APIM Managed Identity to Key Vault (to read
secrets)
resource "azurerm_role_assignment" "apim_key_vault_secret_reader" {
  scope                = var.key_vault_id
  role_definition_name = "Key Vault Secrets User" # Or "Key Vault
Reader" if only reading secret IDs
  principal_id         =
azurerm_api_management_service.apim.identity[0].principal_id
}

# APIM Named Value for OpenAI API Key (linked to Key Vault)
# If using Managed Identity for backend calls, this might not be
strictly needed,
# but can be useful for policies or other integrations.
resource "azurerm_api_management_named_value"
"openai_api_key_named_value" {
  name                = "openai-api-key"
  resource_group_name = var.resource_group_name
  api_management_name = azurerm_api_management_service.apim.name
  display_name        = "OpenAI API Key for AI Services"
  value_from_key_vault {
    secret_id = var.openai_api_key_secret_id
  }
  secret = true
}

# APIM Backend for Azure AI Services OpenAI
resource "azurerm_api_management_backend" "open_ai_backend" {
  name                = "backend-azure-ai-services-openai"
  resource_group_name = var.resource_group_name
  api_management_name = azurerm_api_management_service.apim.name
  protocol            = "http" # Azure OpenAI endpoint is HTTP, APIM
handles TLS
  url                 = var.ai_services_endpoint
  title               = "Azure AI Services OpenAI Backend"

  # If using API Key for backend:
  credentials {
    header {
      name  = "api-key"
      value = "{{openai-api-key}}" # Reference Named Value in APIM
    }
  }

  # If using Managed Identity for backend:
  # credentials {
  #   managed_identity_client_id =
azurerm_api_management_service.apim.identity[0].client_id # Or specify
if user-assigned
  # }
}

# APIM Products
resource "azurerm_api_management_product" "product" {
  for_each = var.apim_products

  product_id            = each.key
  api_management_name   = azurerm_api_management_service.apim.name
  resource_group_name   = var.resource_group_name
  display_name          = each.value.display_name
  description           = each.value.description
  subscription_required = each.value.subscription_required
  approval_required     = each.value.approval_required
  state                 = each.value.state

  tags = {
    environment = var.environment
  }
}

# APIM APIs for each deployed OpenAI model
resource "azurerm_api_management_api" "openai_api" {
  for_each = var.openai_model_deployments_to_expose

  name                = "api-${each.key}"
  resource_group_name = var.resource_group_name
  api_management_name = azurerm_api_management_service.apim.name
  revision            = "1"
  display_name        = "OpenAI - ${each.key}"
  path                = "openai/${each.key}" # e.g., /openai/gpt-35-
```

```
turbo-dev
  protocols           = ["https"]
  service_url         = var.ai_services_endpoint # Important for
swagger generation

  # Associate with the backend
  backend_id          =
azurerm_api_management_backend.open_ai_backend.id

  # Example: If you have a generic OpenAPI spec for Azure OpenAI's
Chat Completions
  # import {
  #   content_format = "openapi-json"
  #   content_value  =
file("${path.module}/swagger/azure_openai_chat_completions.json") #
Needs to exist
  # }

  tags = ["openai", var.environment, each.key]
}

# Link APIs to Products
resource "azurerm_api_management_product_api" "product_api_link" {
  for_each = var.openai_model_deployments_to_expose

  api_name            =
azurerm_api_management_api.openai_api[each.key].name
  product_id          = azurerm_api_management_product.product["open-
ai-models"].product_id
  api_management_name = azurerm_api_management_service.apim.name
  resource_group_name = var.resource_group_name
}

# API Policy for each OpenAI deployment API
# This policy rewrites the URL and injects the API key (if used)
resource "azurerm_api_management_api_policy"
"openai_api_deployment_policy" {
  for_each = var.openai_model_deployments_to_expose

  api_management_name = azurerm_api_management_service.apim.name
  resource_group_name = var.resource_group_name
  api_id              =
azurerm_api_management_api.openai_api[each.key].id

  # Reference local policy file for readability and reusability
  xml_content =
templatefile("${path.module}/policies/openai_rewrite_policy.xml", {
    deployment_name    = each.key,
    openai_api_version = var.openai_api_version
  })
}
```

```
# APIM Diagnostics to Log Analytics
resource "azurerm_monitor_diagnostic_setting" "apim_diagnostics" {
  name                       = "apim-diagnostics"
  target_resource_id         = azurerm_api_management_service.apim.id
  log_analytics_workspace_id = var.log_analytics_workspace_id

  log {
    category = "GatewayLogs"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30
    }
  }
  metric {
    category = "AllMetrics"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30
    }
  }
}

# APIM Integration with Application Insights
resource "azurerm_api_management_logger" "app_insights_logger" {
  name                = "appinsights-logger"
  resource_group_name = var.resource_group_name
  api_management_name = azurerm_api_management_service.apim.name
  application_insights {
    instrumentation_key = var.application_insights_instrumentation_key
  }
}

resource "azurerm_api_management_diagnostic" "apim_diagnostic" {
  api_management_id = azurerm_api_management_service.apim.id
  identifier        = "applicationinsights" # Fixed value for default
App Insights diagnostic
  resource_id       = azurerm_api_management_service.apim.id # It
self-refers to enable logging

  # Ensure the logger is referenced here
  logger_id = azurerm_api_management_logger.app_insights_logger.id

  enabled = true
  sampling_percentage = 100 # Adjust as needed for production
  verbosity           = "verbose" # "verbose", "normal", "error"

  frontend_request {
    body_bytes        = 8192
    headers_to_log = ["Content-Type", "Ocp-Apim-Subscription-Key"]
  }
  frontend_response {
    body_bytes        = 8192
    headers_to_log = ["Content-Type"]
  }
  backend_request {
    body_bytes        = 8192
    headers_to_log = ["Content-Type", "api-key"] # Log API Key in dev
for debug if needed, NOT IN PROD
  }
  backend_response {
    body_bytes        = 8192
    headers_to_log = ["Content-Type"]
  }
}

output "apim_id" {
  value = azurerm_api_management_service.apim.id
}

output "apim_gateway_url" {
  value = azurerm_api_management_service.apim.gateway_url
}

# modules/apim/variables.tf

variable "resource_group_name" {
  description = "The name of the resource group."
  type        = string
}

variable "location" {
  description = "The Azure region."
  type        = string
}

variable "environment" {
  description = "The deployment environment."
  type        = string
}

variable "apim_subnet_id" {
  description = "The ID of the subnet for the APIM instance."
  type        = string
}

variable "apim_sku_name" {
  description = "SKU for API Management (e.g., 'Developer_1',
'Premium_1')."
  type        = string
}

variable "key_vault_id" {
  description = "The ID of the Key Vault for APIM secrets."
  type        = string
}

variable "openai_api_key_secret_id" {
  description = "The ID of the Key Vault secret containing the OpenAI
API Key."
  type        = string
}

variable "ai_services_endpoint" {
  description = "The endpoint URL of the Azure AI Services Account."
  type        = string
}

variable "log_analytics_workspace_id" {
  description = "The ID of the Log Analytics Workspace for APIM
diagnostics."
  type        = string
}

variable "application_insights_id" {
  description = "The ID of the Application Insights resource for
APIM."
  type        = string
}

variable "application_insights_instrumentation_key" {
  description = "The instrumentation key for Application Insights."
  type        = string
  sensitive   = true
}

variable "apim_products" {
  description = "A map of APIM products to create."
  type = map(object({
    display_name = string
    description  = string
    state        = string
    subscription_required = bool
    approval_required     = bool
  }))
}

variable "openai_model_deployments_to_expose" {
  description = "A map of OpenAI model deployments from the AI
Services account to expose via APIM APIs."
  type = map(object({
    model_name    = string
    model_version = string
    capacity      = number
  }))
```

```
}

variable "openai_api_version" {
  description = "The API version to use for Azure OpenAI calls (e.g.,
'2024-02-01')."
  type        = string
}
```

### modules/apim/policies/openai_rewrite_policy.xml

```
<policies>
    <inbound>
        <base />
        <rewrite-uri
template="/openai/deployments/${deployment_name}/chat/completions?api-
version=${openai_api_version}" />

        <set-header name="api-key" exists-action="override">
            <value>{{openai-api-key}}</value>
        </set-header>

        <set-header name="Ocp-Apim-Subscription-Key" exists-
action="delete" />
    </inbound>
    <backend>
        <base />
    </backend>
    <outbound>
        <base />
    </outbound>
    <on-error>
        <base />
    </on-error>
</policies>
```

## 10. modules/cost_management/main.tf

```
# modules/cost_management/main.tf

# Event Hubs Namespace for Cost Data
resource "azurerm_eventhub_namespace" "main" {
  name                = "evhns-${var.environment}-cost"
  location            = var.location
  resource_group_name = var.resource_group_name
  sku                 = "Standard" # Basic or Standard based on
throughput needs
  capacity            = 1          # Throughput units

  tags = {
    environment = var.environment
    project     = "CostManagement"
    service     = "EventHub"
  }
}

# Event Hub for Cost Data
resource "azurerm_eventhub" "cost_data" {
  name                = var.event_hub_name
  namespace_name      = azurerm_eventhub_namespace.main.name
  resource_group_name = var.resource_group_name
  partition_count     = 1 # Or more, depending on expected data volume
  message_retention_in_days = 1 # Or more, depending on processing
frequency

  tags = {
    environment = var.environment
    project     = "CostManagement"
  }
}

# Authorization Rule for Event Hub (for Function App to write, or
Power BI to read)
resource "azurerm_eventhub_authorization_rule" "send_receive_rule" {
  name                = "SendListenRule"
  namespace_name      = azurerm_eventhub_namespace.main.name
  eventhub_name       = azurerm_eventhub.cost_data.name
  resource_group_name = var.resource_group_name
  listen              = true
  send                = true
  manage              = false

  tags = {
    environment = var.environment
  }
}

# Azure Cost Management Export (to send data to Event Hub)
# This will export actual billing data.
# Ensure the identity used by Terraform has 'Cost Management
Contributor' role on the subscription.
resource "azurerm_consumption_budget" "daily_export_budget" { # Using
budget as a placeholder for export, actual export resource differs
  # Note: Azure has a specific
'azurerm_consumption_cost_management_export' resource but it's new.
  # If it's not yet in your provider version, you might need to use a
Function App to pull data.
  # For the purpose of this example, we assume you'd configure an
export via portal or the
  # Consumption API, and this event hub acts as the destination.
  # If azurerm_consumption_cost_management_export exists:
  # resource "azurerm_consumption_cost_management_export"
```

```
"daily_cost_export" {
  #   name                  = "DailyCostExport"
  #   resource_group_name   = var.resource_group_name
  #   scope                 = var.consumption_api_scope
  #   recurrence_period_start_date = "2024-01-01T00:00:00Z" # Adjust
start date
  #   recurrence_period_end_date = "2099-01-01T00:00:00Z"
  #   recurrence_type       = "Daily"
  #   format                = "Csv"
  #   delivery_info {
  #     container_id = azurerm_storage_account.cost_export_storage.id
# Assuming dedicated storage
  #     root_folder_path = "costexports"
  #   }
  #   tags = {
  #     environment = var.environment
  #   }
  # }
  #
  # If this resource is not available, you would manually set up the
export in Azure,
  # or write a Function App to pull data from the Consumption API and
push to Event Hub.

  # Placeholder for budget - you'd likely set up actual budget alerts
here
  # This resource is just a placeholder to acknowledge cost management
setup.
  amount            = 1000.0 # Example budget amount
  time_grain        = "Monthly"
  time_period_start_date = "2024-01-01T00:00:00Z"
  scope             = var.consumption_api_scope
  name              = "budget-${var.environment}-aihub"
  resource_group_name = var.resource_group_name

  tags = {
    environment = var.environment
  }
}

# Diagnostics for Event Hub Namespace
resource "azurerm_monitor_diagnostic_setting" "eventhub_diagnostics" {
  name                       = "eventhub-diagnostics"
  target_resource_id         = azurerm_eventhub_namespace.main.id
  log_analytics_workspace_id = var.log_analytics_workspace_id

  log {
    category = "ArchiveLogs"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30
    }
  }
  log {
    category = "OperationalLogs"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30
    }
  }
  metric {
    category = "AllMetrics"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30
    }
  }
}

output "event_hub_namespace_id" {
  value = azurerm_eventhub_namespace.main.id
}

output "event_hub_auth_rule_primary_connection_string" {
  value     =
azurerm_eventhub_authorization_rule.send_receive_rule.primary_connecti
on_string
  sensitive = true
}

output "event_hub_name" {
  value = azurerm_eventhub.cost_data.name
}

# modules/cost_management/variables.tf

variable "resource_group_name" {
  description = "The name of the resource group."
  type        = string
}

variable "location" {
  description = "The Azure region."
  type        = string
}

variable "environment" {
  description = "The deployment environment."
  type        = string
}
```

```
variable "event_hub_namespace_name" {
  description = "Name for the Event Hub Namespace for cost data."
  type        = string
}

variable "event_hub_name" {
  description = "Name for the Event Hub for cost data."
  type        = string
}

variable "consumption_api_scope" {
  description = "The scope for the Azure Consumption API export (e.g.,
'/subscriptions/YOUR_SUB_ID')."
  type        = string
}

variable "log_analytics_workspace_id" {
  description = "The ID of the Log Analytics Workspace for
diagnostics."
  type        = string
}
```

## 11. modules/function_app_cost_processor/main.tf

```
# modules/function_app_cost_processor/main.tf

# Storage account for Function App code and state (separate from
general_purpose_storage)
resource "azurerm_storage_account" "function_storage" {
  name                     = "stfunc${var.environment}costproc" # Max
24 chars, lowercase
  resource_group_name      = var.resource_group_name
  location                 = var.location
  account_tier             = "Standard"
  account_replication_type = var.function_app_storage_sku # e.g.,
"LRS"

  # Private Endpoint for Function App Storage
  network_rules {
    default_action = "Deny"
    ip_rules       = []
    virtual_network_subnet_ids = [var.functions_subnet_id] # Allow
function app VNet access
    bypass         = ["AzureServices"]
  }

  tags = {
    environment = var.environment
    project     = "CostManagement"
    service     = "FunctionAppStorage"
  }
}

resource "azurerm_private_endpoint" "function_storage_pe" {
  name                = "pe-
${azurerm_storage_account.function_storage.name}"
  location            = var.location
  resource_group_name = var.resource_group_name
  subnet_id           = var.functions_subnet_id

  private_service_connection {
    name                           = "psc-
${azurerm_storage_account.function_storage.name}"
    is_manual_connection           = false
    private_connection_resource_id =
azurerm_storage_account.function_storage.id
    subresource_names              = ["blob"] # Or "file", "queue",
"table" as needed
  }
}

# Private DNS Zone and Link for Function App Storage
resource "azurerm_private_dns_zone" "function_storage_dns_zone" {
  name                = "privatelink.blob.core.windows.net" # Standard
zone for blob storage
  resource_group_name = var.resource_group_name
}

resource "azurerm_private_dns_zone_virtual_network_link"
"function_storage_dns_link" {
  name                  = "link-
${azurerm_storage_account.function_storage.name}"
  resource_group_name   = var.resource_group_name
  private_dns_zone_name =
azurerm_private_dns_zone.function_storage_dns_zone.name
  virtual_network_id    = var.vnet_id
  registration_enabled  = false
}

resource "azurerm_private_dns_a_record" "function_storage_a_record" {
  name                = azurerm_storage_account.function_storage.name
  zone_name           =
azurerm_private_dns_zone.function_storage_dns_zone.name
  resource_group_name = var.resource_group_name
  ttl                 = 300
  records             =
azurerm_private_endpoint.function_storage_pe.private_service_connectio
n[0].private_ip_address
}


resource "azurerm_app_service_plan" "main" {
  name                = "plan-${var.function_app_name}"
  location            = var.location
  resource_group_name = var.resource_group_name
  kind                = "FunctionApp"
  sku {
    tier = "ElasticPremium" # Recommended for VNet integration and
production workloads
    size = "EP1"
  }
  # Assign to the subnet if VNet integration needed:
  # If using Azure Functions Premium plan, VNet integration happens at
the plan level
  # virtual_network_subnet_id = var.functions_subnet_id
  # outbound_type = "VnetInjection" # Required for private access to
backends in VNet
}

resource "azurerm_linux_function_app" "main" { # Or
azurerm_function_app for Windows
  name                       = var.function_app_name
  location                   = var.location
  resource_group_name        = var.resource_group_name
  service_plan_id            = azurerm_app_service_plan.main.id
  storage_account_name       =
azurerm_storage_account.function_storage.name
  storage_account_access_key =
azurerm_storage_account.function_storage.primary_access_key
  os_type                    = "Linux" # Or "Windows"
  public_network_access_enabled = false # Restrict public access,
force via VNet/Private Endpoint

  site_config {
    application_insights_connection_string =
var.application_insights_connection_string
    application_insights_key               =
var.application_insights_instrumentation_key
    always_on                              = true
    ip_restriction {
      ip_address        = "0.0.0.0/0"
      virtual_network_subnet_id = var.functions_subnet_id # Allow
traffic from its own VNet
      action            = "Allow"
      priority          = 100
      name              = "VNetIntegration"
    }
  }

  app_settings = {
    "FUNCTIONS_WORKER_RUNTIME" = "python"
    "PYTHON_VERSION"           = var.function_app_python_version
    "EventHubConnection"       =
"Endpoint=sb://${var.event_hub_namespace_name}.servicebus.windows.net/
;SharedAccessKeyName=${azurerm_eventhub_authorization_rule.event_hub_r
eceive_rule.name};SharedAccessKey=${azurerm_eventhub_authorization_rul
e.event_hub_receive_rule.primary_key};EntityPath=${var.event_hub_name}
"
    "EventHubName"             = var.event_hub_name
    "AzureWebJobsStorage"      =
azurerm_storage_account.function_storage.primary_connection_string
    "WEBSITE_VNET_ROUTE_ALL"   = "1" # Force all outbound traffic
through VNet (for private endpoints)
  }

  identity {
    type = "SystemAssigned" # For accessing Event Hub, Storage, etc.
  }

  tags = {
    environment = var.environment
    project     = "CostManagement"
    service     = "FunctionApp"
  }
}

# Role Assignment for Function App Managed Identity to Event Hub (to
read cost data)
resource "azurerm_role_assignment" "function_app_eventhub_reader" {
  scope                = var.event_hub_namespace_id
  role_definition_name = "Azure Event Hubs Data Receiver"
  principal_id         =
azurerm_linux_function_app.main.identity[0].principal_id
}

# Deploy Function App code (Python example)
resource "azurerm_zip_blob" "function_app_code" {
  name                   = "${var.function_app_name}.zip"
  storage_account_name   =
azurerm_storage_account.function_storage.name
  storage_container_name = "$web" # Special container for static sites
/ function app deployment from zip
  type                   = "Block"
  source_content         =
filebase64sha256("${path.module}/src/function_app.zip") # Assume you
zip your function app code

  depends_on = [
    azurerm_linux_function_app.main
  ]
}

# Authorization rule for the Function App to *receive* from Event Hub
# This should be dedicated to the Function App for least privilege.
resource "azurerm_eventhub_authorization_rule"
"event_hub_receive_rule" {
  name                = "ReceiveRuleFuncApp"
  namespace_name      = var.event_hub_namespace_name
  eventhub_name       = var.event_hub_name
  resource_group_name = var.resource_group_name
  listen              = true
  send                = false
  manage              = false

  tags = {
    environment = var.environment
  }
}

# Diagnostics for Function App
resource "azurerm_monitor_diagnostic_setting"
"function_app_diagnostics" {
  name                       = "funcapp-diagnostics"
  target_resource_id         = azurerm_linux_function_app.main.id
  log_analytics_workspace_id = var.log_analytics_workspace_id

  log {
    category = "FunctionAppLogs"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30
    }
  }
  metric {
    category = "AllMetrics"
    enabled  = true
    retention_policy {
      enabled = true
      days    = 30
    }
  }
}

output "function_app_id" {
  value = azurerm_linux_function_app.main.id
}

# modules/function_app_cost_processor/variables.tf

variable "resource_group_name" {
  description = "The name of the resource group."
  type        = string
}

variable "location" {
  description = "The Azure region."
  type        = string
```

```
}

variable "environment" {
  description = "The deployment environment."
  type        = string
}

variable "function_app_name" {
  description = "Name of the Function App."
  type        = string
}

variable "function_app_storage_sku" {
  description = "SKU for the Function App storage account."
  type        = string
}

variable "function_app_python_version" {
  description = "Python version for the Function App."
  type        = string
}

variable "functions_subnet_id" {
  description = "The ID of the subnet for the Function App's VNet
integration/Private Endpoint."
  type        = string
}

variable "event_hub_namespace_id" {
  description = "The ID of the Event Hub Namespace."
  type        = string
}

variable "event_hub_name" {
  description = "The name of the Event Hub for cost data."
  type        = string
}

variable "log_analytics_workspace_id" {
  description = "The ID of the Log Analytics Workspace for Function
App diagnostics."
  type        = string
}

variable "application_insights_connection_string" {
  description = "The connection string for Application Insights."
  type        = string
  sensitive   = true
}

variable "application_insights_instrumentation_key" {
  description = "The instrumentation key for Application Insights."
  type        = string
  sensitive   = true
}

variable "vnet_id" {
  description = "The ID of the Virtual Network where the Function App
subnet resides."
  type        = string
}
```

### modules/function_app_cost_processor/src/function_app.py (Example Python Function App)

This is a basic example. You would develop your actual cost processing logic here.

```
# modules/function_app_cost_processor/src/function_app.py
import azure.functions as func
import logging
import json
import os

app = func.FunctionApp(http_auth_level=func.AuthLevel.FUNCTION)

# Event Hub Trigger example
@app.event_hub_message_trigger(arg_name="azeventhub",
event_hub_name="costdata", connection="EventHubConnection")
def EventHubProcessor(azeventhub: func.EventHubEvent):
    logging.info('Python EventHub trigger processed an event.')

    for event in azeventhub:
        try:
            event_body = event.get_body().decode('utf-8')
            cost_data = json.loads(event_body)
            logging.info(f"Received cost data: {cost_data}")

            # --- Your cost processing logic goes here ---
            # e.g., Transform data, store in Azure SQL, Data Lake, or
push to Power BI streaming dataset
            # For Power BI streaming, you'd typically make an HTTP
POST request to the Power BI API.
            # You'd need to handle authentication (e.g., Azure AD app
registration with Power BI permissions).
            # Example (conceptual):
            # import requests
            # power_bi_url = os.environ.get("PowerBIStreamingUrl")
            # if power_bi_url:
            #     requests.post(power_bi_url, json=cost_data)
            # -------------------------------------------

            logging.info(f"Processed event: {event.sequence_number}")
        except Exception as e:
            logging.error(f"Error processing event: {e}")

# You'll also need host.json and requirements.txt for the Function App
# host.json:
# {
#   "version": "2.0",
#   "logging": {
#     "applicationInsights": {
#       "samplingSettings": {
#         "isEnabled": true,
#         "excludedTypes": "Request"
#       }
#     }
#   },
#   "extensionBundle": {
#     "id": "Microsoft.Azure.Functions.ExtensionBundle",
#     "version": "[4.*, 5.0.0)"
#   }
# }

# requirements.txt:
# azure-functions
# azure-eventhub
# azure-identity # If you use Managed Identity to access other Azure
services from function
# requests # If you make HTTP calls (e.g., to Power BI)
```

You would then zip this src folder (including host.json, requirements.txt) into function_app.zip and place it in modules/function_app_cost_processor/src/ for Terraform to deploy.

## Deployment Workflow (CI/CD using Azure DevOps)

1. **Repository Setup:** All this code would be in a Git repository.
2. **Service Principal:** Create an Azure Service Principal with Contributor permissions on the target subscription (or more granular RBAC roles for production) and Reader on the VNet resource group.
3. **Azure DevOps Pipeline:**
    - **Variables Group:** Create a variable group for each environment (dev, prod). Store sensitive variables (like arm_subscription_id, openai_api_key if not pulling from KV directly, application_insights_instrumentation_key) securely in these groups.
    - **Stages:** Define stages for dev, prod. Each stage would depend on the previous one (e.g., prod depends on dev approval).
    - **Jobs:**
        - **terraform init**: Configure backend, initialize providers.
        - **terraform validate**: Check syntax.
        - **terraform plan**: Generate execution plan, save to artifact.
        - **Manual Approval (for prod)**: Review the plan.
        - **terraform apply**: Apply the plan.
    - **Tasks:** Use the AzureResourceManager@3 task with connectedServiceNameARM pointing to your Azure Service Principal connection.
    - **Managed Identity for APIM/Functions:** Ensure your APIM and Function Apps are

deployed first, then grant their Managed Identities the necessary RBAC roles (e.g., "Cognitive Services OpenAI User" on AI Services, "Azure Event Hubs Data Receiver" on Event Hub). This is handled by Terraform's azurerm_role_assignment resources.

   - **Function App Code Deployment:** The azurerm_zip_blob resource handles deploying your zipped Function App code.
   - **terraform fmt and terraform taint (Optional but good for pipelines):** Add terraform fmt -check -recursive and consider terraform taint for specific resource re-creation if needed for hotfixes.

This full code provides a solid foundation for your production-ready, reusable Azure AI Hub accelerator. Remember to **customize all placeholder values** (e.g., subscription IDs, API keys, names) and thoroughly test in your dev environment before promoting to prod.