This is an excellent and common scenario for leveraging Terraform in an enterprise environment. You're aiming for a modular, flexible, and environment-agnostic infrastructure as code (IaC) setup, while integrating with existing resources and preparing for future self-service. Given your specific requirements, especially the need to onboard existing spoke, subscription, VNet, and subnets, and the flexibility to exclude/inject resources, we'll design a modular Terraform structure.

Key Principles for this Solution:

1. **Modularity:** Break down your infrastructure into logical, reusable modules (e.g., apim, monitoring, aihub). This makes the code easier to manage, test, and reuse across different environments.
2. **Environment Agnosticism:** Use variables extensively to parameterize environment-specific values (e.g., resource group names, locations, VNet/subnet IDs).
3. **Conditional Resource Creation:** Utilize count and for_each meta-arguments, along with conditional expressions, to control whether a resource is created or if an existing one is referenced.
4. **Data Sources for Existing Resources:** Leverage Terraform data sources to fetch information about pre-existing Azure resources (VNet, subnets, etc.) without managing them in your Terraform state.
5. **Policy and Product Management in APIM:** Integrate custom policies and products within your APIM module.
6. **Cost Management & Monitoring:** Include modules for Log Analytics, App Insights, Event Hubs, and potentially Function Apps for cost processing.
7. **Azure Pipelines Integration:** Structure the code for easy consumption by Azure DevOps pipelines.
8. **Outputting Key Information:** Output necessary resource IDs or connection strings that might be needed by application teams for their self-service onboarding.

Project Structure (Terraform Modules)

Here's a suggested project structure. Each folder represents a Terraform module.

```
.
├── main.tf
├── variables.tf
├── outputs.tf
├── environments/
│   ├── dev/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── backend.tf
│   ├── qa/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── backend.tf
│   └── prod/
│       ├── main.tf
│       ├── variables.tf
│       └── backend.tf
├── modules/
│   ├── networking/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── apim/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   ├── outputs.tf
│   │   └── policies/ # Directory for XML policy files
│   │       ├── global.xml
│   │       ├── product_policy.xml
│   │       └── api_policy.xml
│   ├── aihub/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── monitoring/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── cost_management/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   └── shared_services/ # For resources like Key Vault, Storage
Accounts that might be shared across modules
│       ├── main.tf
│       ├── variables.tf
│       └── outputs.tf
└── azure-pipelines.yml
```

Detailed Terraform Code Structure

Let's break down the content of each file.

1. Root (main.tf, variables.tf, outputs.tf)

These files at the root of your repository will define the Azure provider, backend configuration (typically for remote state), and call the environment-specific configurations.

main.tf (Root)

```
# main.tf (Root) - This file will primarily call the environment-
specific configurations.
# It's good practice to keep this minimal and delegate to environment
folders.

terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "~> 3.0"
    }
    random = {  # Used for generating unique names if needed
      source = "hashicorp/random"
      version = "~> 3.0"
    }
  }
}

# Azure Provider Configuration
provider "azurerm" {
  features {}
}

# Backend configuration (e.g., Azure Storage Account for remote state)
# This will be overridden in environment-specific backend.tf files.
# It's here for reference but the actual config will be in
environments/*/backend.tf
# terraform {
#   backend "azurerm" {
#     resource_group_name  = "tfstate-rg"
#     storage_account_name = "tfstatesa"
#     container_name       = "tfstate"
#     key                  = "aihub-accelerator.tfstate"
#   }
# }

# No direct resource deployment here. All deployments happen through
environment folders.
```

variables.tf (Root)

```
# variables.tf (Root) - Defines common variables used across
environments or to set defaults.

variable "resource_group_name_prefix" {
  description = "Prefix for resource group names. Will be suffixed
with environment name."
  type        = string
  default     = "rg-aihub"
}

variable "location" {
  description = "Azure region where resources will be deployed."
  type        = string
  default     = "East US" # Example default, override in envts
}

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

outputs.tf (Root)

```
# outputs.tf (Root) - Outputs from the overall deployment.
# Example: Outputting the APIM gateway URL
# This will be populated from the specific environment's main.tf
output.
output "apim_gateway_url" {
  description = "The URL of the API Management gateway."
  value       = "Consult environment-specific outputs (e.g.,
environments/dev/outputs.tf)"
}

# Example: Outputting AI Hub endpoint
output "ai_hub_endpoint" {
  description = "The endpoint for the Azure AI Hub."
  value       = "Consult environment-specific outputs (e.g.,
environments/dev/outputs.tf)"
}
```

2. Environments (environments/dev, qa, prod)

Each environment will have its own main.tf, variables.tf, and backend.tf.
environments/dev/main.tf

```
# environments/dev/main.tf

# Inherit root variables
variable "resource_group_name_prefix" {}
variable "location" {}
variable "environment" {}

# Remote backend configuration for dev environment
terraform {
  backend "azurerm" {
    resource_group_name  = "tfstate-rg-dev" # Unique RG for dev
tfstate
    storage_account_name = "tfstatedevaihub" # Unique SA for dev
tfstate
    container_name       = "tfstate"
    key                  = "aihub-accelerator-dev.tfstate"
    subscription_id      = "YOUR_DEV_SUBSCRIPTION_ID" # Replace with
your Dev Subscription ID
  }
}

# Data source for existing VNet and Subnets
# This is crucial for onboarding to existing network infrastructure.
data "azurerm_resource_group" "spoke_rg" {
  name = var.existing_spoke_resource_group_name
}

data "azurerm_virtual_network" "spoke_vnet" {
  name                = var.existing_vnet_name
  resource_group_name = data.azurerm_resource_group.spoke_rg.name
```

```
}

data "azurerm_subnet" "apim_subnet" {
  name                 = var.existing_apim_subnet_name
  virtual_network_name = data.azurerm_virtual_network.spoke_vnet.name
  resource_group_name  =
data.azurerm_virtual_network.spoke_vnet.resource_group_name
}

data "azurerm_subnet" "aihub_subnet" {
  name                 = var.existing_aihub_subnet_name
  virtual_network_name = data.azurerm_virtual_network.spoke_vnet.name
  resource_group_name  =
data.azurerm_virtual_network.spoke_vnet.resource_group_name
}

# Create a resource group for this environment's new resources
resource "azurerm_resource_group" "aihub_rg" {
  name     = "${var.resource_group_name_prefix}-${var.environment}"
  location = var.location
}

# ----------------------------------------------------------------------
-------------------
# Module Calls
# ----------------------------------------------------------------------
-------------------

# Monitoring Module
module "monitoring" {
  source = "../../modules/monitoring" # Relative path to the module

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

  # Conditional creation/injection for Log Analytics Workspace
  create_log_analytics_workspace =
var.create_new_log_analytics_workspace
  existing_log_analytics_workspace_id =
var.existing_log_analytics_workspace_id
  log_analytics_workspace_name = var.log_analytics_workspace_name
}

# Shared Services Module (e.g., Key Vault for APIM secrets, Storage
for functions)
module "shared_services" {
  source = "../../modules/shared_services"

  resource_group_name = azurerm_resource_group.aihub_rg.name
  location            = var.location
  environment         = var.environment
  log_analytics_workspace_id =
module.monitoring.log_analytics_workspace_id
}

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

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

  # Pass existing subnet ID to APIM module
  apim_subnet_id = data.azurerm_subnet.apim_subnet.id

  # Dependency on shared_services for Key Vault/Storage for
policies/backends
  key_vault_id = module.shared_services.key_vault_id
  log_analytics_workspace_id =
module.monitoring.log_analytics_workspace_id

  # Example of passing custom policies/products dynamically
  custom_policies_dir = "../../modules/apim/policies"
  apim_products       = var.apim_products
  open_ai_backends    = var.open_ai_backends # For onboarding OpenAI
models as backends
}

# AI Hub Module
module "aihub" {
  source = "../../modules/aihub"

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

  # Pass existing subnet ID to AI Hub module (if required for Private
Endpoints)
  aihub_subnet_id = data.azurerm_subnet.aihub_subnet.id

  # Integrate with monitoring
  log_analytics_workspace_id =
module.monitoring.log_analytics_workspace_id
}

# Cost Management Module
module "cost_management" {
  source = "../../modules/cost_management"

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

  # Integrate with monitoring components
  log_analytics_workspace_id =
module.monitoring.log_analytics_workspace_id
  event_hub_namespace_id     =
module.shared_services.event_hub_namespace_id # Assuming Event Hub is
in shared_services
  # Potentially pass storage account for Function App code or Power BI
output
  storage_account_id         =
module.shared_services.storage_account_id
}

# Outputs from this environment
output "apim_gateway_url" {
  description = "The URL of the API Management gateway for
${var.environment}."
  value       = module.apim.apim_gateway_url
}

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

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

environments/dev/variables.tf

```
# environments/dev/variables.tf

# General environment variables
variable "existing_spoke_resource_group_name" {
  description = "Name of the existing spoke resource group where VNet
resides."
  type        = string
  default     = "your-dev-spoke-rg" # CHANGE THIS
}

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

variable "existing_apim_subnet_name" {
  description = "Name of the existing subnet for APIM."
  type        = string
  default     = "subnet-apim-dev" # CHANGE THIS
}

variable "existing_aihub_subnet_name" {
  description = "Name of the existing subnet for AI Hub (e.g., for
private endpoints)."
  type        = string
  default     = "subnet-aihub-dev" # CHANGE THIS
}

# Monitoring specific variables for conditional creation/injection
variable "create_new_log_analytics_workspace" {
  description = "Set to true to create a new Log Analytics Workspace,
false to use an existing one."
  type        = bool
  default     = true
}

variable "existing_log_analytics_workspace_id" {
  description = "ID of an existing Log Analytics Workspace (if
create_new_log_analytics_workspace is false)."
  type        = string
  default     = null
}

variable "log_analytics_workspace_name" {
  description = "Name for the new Log Analytics Workspace (if created)
or the existing one."
  type        = string
  default     = "law-aihub-dev"
}

# APIM specific variables
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
  }))
  default = {
    "open-ai-models" = {
      display_name = "OpenAI Models"
      description  = "Access to various OpenAI models."
      state        = "published"
      subscription_required = true
      approval_required     = true
    }
  }
}

variable "open_ai_backends" {
  description = "A map of OpenAI service backends to onboard to APIM."
  type = map(object({
    display_name = string
    url          = string
    protocol     = string
  }))
  default = {
    "openai-gpt4o" = {
      display_name = "Azure OpenAI GPT-4o"
      url          =
"https://yourazureopenai.openai.azure.com/openai/deployments/gpt-4o" #
CHANGE THIS
      protocol     = "http"
    }
    "openai-gpt35turbo" = {
      display_name = "Azure OpenAI GPT-3.5 Turbo"
      url          =
"https://yourazureopenai.openai.azure.com/openai/deployments/gpt-35-
turbo" # CHANGE THIS
      protocol     = "http"
    }
  }
}

# Add other environment-specific variables here
# For prod, you might have different scale units, SKUs, or additional
security configurations.
```

environments/dev/backend.tf (for remote state)

```
# environments/dev/backend.tf
terraform {
  backend "azurerm" {
    resource_group_name  = "tfstate-rg-dev" # This RG must exist prior
to first `terraform init`
    storage_account_name = "tfstatedevaihub" # This SA must exist
prior to first `terraform init`
    container_name       = "tfstate"
    key                  = "aihub-accelerator-dev.tfstate"
    subscription_id      = "YOUR_DEV_SUBSCRIPTION_ID" # Replace with
your Dev Subscription ID
  }
}
```

Note: Repeat the main.tf and variables.tf structure for qa and prod environments, adjusting default values and backend configurations as needed. For prod, you'll likely use higher-tier SKUs and more robust settings. The backend.tf for each environment *must* point to a unique storage account or key to prevent state conflicts.

3. Modules (modules/)

modules/networking (Optional - for new VNets, but you have existing)

Given you have existing VNet/subnets, this module might not be strictly necessary for your initial deployment. However, it's good to keep in mind if you ever need to provision new networks with Terraform. The data sources in environments/*/main.tf handle the existing network.

modules/apim

```
# modules/apim/main.tf
resource "azurerm_resource_group" "apim_rg" {
  name     = var.resource_group_name
  location = var.location
  # lifecycle { ignore_changes = [name] } # If you want to keep the
resource group outside this module's management
}

resource "azurerm_api_management_service" "apim" {
  name                = "apim-${var.environment}-aihub"
  location            = azurerm_resource_group.apim_rg.location
  resource_group_name = azurerm_resource_group.apim_rg.name
  publisher_name      = "AI Hub Team"
  publisher_email     = "aihub@example.com"
  sku_name            = "Developer_1" # Adjust for Prod (e.g.,
"Premium_1" or "Premium_2")

  virtual_network_type = "External" # Or "Internal" if you want to
expose only within VNet
  virtual_network_configuration {
    subnet_id = var.apim_subnet_id
  }

  hostname_configuration {
    proxy {
      default_ssl_binding = true
      hostname            = "api.${var.environment}.aihub.com" #
Custom domain, if any
    }
  }

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

# Configure APIM Logger to Log Analytics Workspace
resource "azurerm_api_management_logger" "apim_logger" {
  name                      = "aihub-apim-logger"
  api_management_name       = azurerm_api_management_service.apim.name
  resource_group_name      = azurerm_resource_group.apim_rg.name
  application_insights_id  = var.application_insights_id # Assumes App
Insights is provisioned by monitoring module
  resource_id              = var.log_analytics_workspace_id # For
sending logs to LAW
  is_buffered              = true # Enable for performance
}

# Global APIM Policy
resource "azurerm_api_management_api_policy" "global_policy" {
  api_management_name = azurerm_api_management_service.apim.name
  resource_group_name = azurerm_resource_group.apim_rg.name
  api_id              = azurerm_api_management_service.apim.id #
Applies to all APIs

  xml_content = file("${path.module}/policies/global.xml")
}

# 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   = azurerm_resource_group.apim_rg.name
  display_name          = each.value.display_name
  description           = each.value.description
  subscription_required = each.value.subscription_required
  approval_required     = each.value.approval_required
  published             = each.value.state == "published"
  # terms = "Your terms of use for this product." # Optional
}

# APIM Backends (for OpenAI models)
resource "azurerm_api_management_backend" "open_ai_backend" {
  for_each = var.open_ai_backends

  name                = "backend-${each.key}"
  resource_group_name = azurerm_resource_group.apim_rg.name
  api_management_name = azurerm_api_management_service.apim.name
  protocol            = each.value.protocol
  url                 = each.value.url
  title               = each.value.display_name

  # Example: Headers for OpenAI authentication
  proxy {
    url = each.value.url
  }

  credentials {
    header {
      name  = "api-key"
      value = module.shared_services.openai_api_key_secret_id # Get
API Key from Key Vault
    }
  }
}

# APIM APIs (e.g., to expose specific OpenAI models)
resource "azurerm_api_management_api" "openai_api" {
  for_each = var.open_ai_backends # Creating an API for each backend

  name                = "api-${each.key}"
  resource_group_name = azurerm_resource_group.apim_rg.name
  api_management_name = azurerm_api_management_service.apim.name
  revision            = "1"
  display_name        = each.value.display_name
  path                = each.key
  protocols           = ["https"]

  # Link to the backend
  backend_id          =
azurerm_api_management_backend.open_ai_backend[each.key].id

  # Example: Import a OpenAPI specification if you have one for the
OpenAI service
  # This makes it easier to define operations.
  # import {
  #   content_format = "openapi"
  #   content_value  = file("path/to/openai_spec.json")
  # }

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

# Link APIs to Products (e.g., all OpenAI APIs to the "open-ai-models"
product)
resource "azurerm_api_management_product_api" "product_api_link" {
  for_each = { for k, v in var.open_ai_backends : k => v } # Iterate
over open_ai_backends

  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 = azurerm_resource_group.apim_rg.name
}

# Example API Policy (for a specific API)
resource "azurerm_api_management_api_policy" "openai_api_policy" {
  for_each = var.open_ai_backends

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

  xml_content = file("${path.module}/policies/api_policy.xml") #
Example API specific policy
}

# Policy for a product
resource "azurerm_api_management_product_policy"
"openai_product_policy" {
  product_id          = azurerm_api_management_product.product["open-
ai-models"].product_id
  api_management_name = azurerm_api_management_service.apim.name
  resource_group_name = azurerm_resource_group.apim_rg.name

  xml_content = file("${path.module}/policies/product_policy.xml")
}

output "apim_gateway_url" {
  value = azurerm_api_management_service.apim.gateway_url
  description = "The gateway URL for the API Management service."
}

output "apim_developer_portal_url" {
  value = azurerm_api_management_service.apim.developer_portal_url
  description = "The URL for the API Management developer portal."
}
```

modules/apim/variables.tf

```
# modules/apim/variables.tf
variable "resource_group_name" {
  description = "The name of the resource group where APIM will be
deployed."
  type        = string
}

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

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

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

```
variable "key_vault_id" {
  description = "The ID of the Key Vault for APIM secrets (e.g.,
OpenAI API key)."
  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 instance for APIM
diagnostics."
  type        = string
  default = null # Can be null if App Insights is not used for APIM
logging
}

variable "custom_policies_dir" {
  description = "Directory containing custom APIM policy XML files."
  type        = string
  default     = "./policies"
}

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 "open_ai_backends" {
  description = "A map of OpenAI service backends to onboard to APIM."
  type = map(object({
    display_name = string
    url          = string
    protocol     = string
  }))
}
```

modules/apim/outputs.tf

```
# modules/apim/outputs.tf
output "apim_gateway_url" {
  value = azurerm_api_management_service.apim.gateway_url
  description = "The gateway URL for the API Management service."
}

output "apim_id" {
  value = azurerm_api_management_service.apim.id
  description = "The ID of the API Management service."
}
```

modules/apim/policies/global.xml (Example)

```
<policies>
    <inbound>
        <base />
        <set-header name="X-Request-ID" exists-action="override">
            <value>@{context.RequestId}</value>
        </set-header>
        </inbound>
    <backend>
        <base />
    </backend>
    <outbound>
        <base />
        </outbound>
    <on-error>
        <base />
    </on-error>
</policies>
```

modules/apim/policies/product_policy.xml (Example)

```
<policies>
    <inbound>
        <base />
        <rate-limit-by-key calls="1000" renewal-period="3600" counter-key="@(context.Subscription.Id)" />
    </inbound>
    <backend>
        <base />
    </backend>
    <outbound>
        <base />
    </outbound>
    <on-error>
        <base />
    </on-error>
</policies>
```

modules/apim/policies/api_policy.xml (Example - for OpenAI API)

```
<policies>
    <inbound>
        <base />
        <rewrite-uri template="/openai/deployments/gpt-4o/chat/completions?api-version=2024-02-01" />
        <set-header name="api-key" exists-action="override">
            <value>{{openai-api-key}}</value>
        </set-header>
    </inbound>
    <backend>
        <base />
    </backend>
    <outbound>
        <base />
    </outbound>
    <on-error>
        <base />
    </on-error>
</policies>
```

Note: The {{openai-api-key}} in the XML policy is a named value in APIM. You would create this named value and link it to a Key Vault secret. Terraform can manage APIM Named Values:

```
resource "azurerm_api_management_named_value"
"openai_api_key_named_value" {
  name                = "openai-api-key"
  resource_group_name = azurerm_resource_group.apim_rg.name
  api_management_name = azurerm_api_management_service.apim.name
  display_name        = "OpenAI API Key"
  value_from_key_vault {
    secret_id = module.shared_services.openai_api_key_secret_id # From
your Key Vault module
  }
  secret = true
}
```

This named value would then be referenced in your policies as {{openai-api-key}}.

modules/aihub

This module would contain the Azure AI Hub resources. The specific resources will depend on what "Azure AI Hub" encompasses for your organization (e.g., Azure Machine Learning Workspace, Azure OpenAI Service, Azure AI Search, etc.).

```
# modules/aihub/main.tf
resource "azurerm_resource_group" "aihub_rg" {
  name     = var.resource_group_name
  location = var.location
}

# Example: Azure Machine Learning Workspace
resource "azurerm_machine_learning_workspace" "ai_workspace" {
  name                          = "mlw-${var.environment}-aihub"
  location                      =
azurerm_resource_group.aihub_rg.location
  resource_group_name           = azurerm_resource_group.aihub_rg.name
  application_insights_id       = var.application_insights_id # From
monitoring module
```

```
  key_vault_id                  = var.key_vault_id            # From
shared_services module
  storage_account_id            = var.storage_account_id      # From
shared_services module
  identity {
    type = "SystemAssigned"
  }
  public_network_access_enabled = false # Ensure private access
  vnet_content_by_default_enabled = true # Allow VNet isolation
  tags = {
    environment = var.environment
    project     = "AIHub"
  }
}

# Example: Azure OpenAI Service
resource "azurerm_cognitive_account" "azure_openai" {
  name                = "aoai-${var.environment}-aihub"
  location            = azurerm_resource_group.aihub_rg.location
  resource_group_name = azurerm_resource_group.aihub_rg.name
  kind                = "OpenAI"
  sku_name            = "S0" # Adjust as needed

  # Private Endpoint connection if you want to lock down access to the
VNet
  private_endpoint_connection {
    name                           = "pe-aoai-${var.environment}"
    subresource_name               = "account"
    is_manual_connection           = false
    virtual_network_id             = var.existing_vnet_id # From
environment's data source
    subnet_id                      = var.aihub_subnet_id
    private_ip_address             = null # Azure assigns
  }

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

# Example: Deploy a model to Azure OpenAI (e.g., GPT-4)
resource "azurerm_cognitive_deployment" "gpt4_deployment" {
  name                 = "gpt-4"
  cognitive_account_id = azurerm_cognitive_account.azure_openai.id
  model {
    format  = "OpenAI"
    name    = "gpt-4"
    version = "0613" # Or latest stable version
  }
  scale {
    type    = "Standard"
    capacity = 1 # Adjust capacity based on expected load
  }
}

output "ai_hub_endpoint" {
  value = azurerm_machine_learning_workspace.ai_workspace.id # Or the
Azure OpenAI endpoint
  description = "The ID/endpoint of the Azure AI Hub main workspace."
}
```

modules/aihub/variables.tf

```
# 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 "aihub_subnet_id" {
  description = "The ID of the existing subnet for AI Hub private
endpoints."
  type        = string
}

variable "existing_vnet_id" {
  description = "The ID of the existing VNet."
  type        = string
}

variable "application_insights_id" {
  description = "The ID of the Application Insights instance for AI
Hub diagnostics."
  type        = string
}

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

variable "storage_account_id" {
  description = "The ID of the Storage Account for AI Hub data."
  type        = string
}

# Add variables for specific AI models, deployments, etc.
```

modules/monitoring

```
# modules/monitoring/main.tf
resource "azurerm_resource_group" "monitoring_rg" {
  name     = var.resource_group_name
  location = var.location
}

# Log Analytics Workspace (conditional creation/injection)
resource "azurerm_log_analytics_workspace" "main" {
  count = var.create_log_analytics_workspace ? 1 : 0 # Conditionally
create
  name                = var.log_analytics_workspace_name
  location            = azurerm_resource_group.monitoring_rg.location
  resource_group_name = azurerm_resource_group.monitoring_rg.name
  sku                 = "PerGB2018" # Or "Consumption"
  retention_in_days   = 30 # Adjust as per retention policy

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

# Data source for existing Log Analytics Workspace
data "azurerm_log_analytics_workspace" "existing" {
  count = var.create_log_analytics_workspace ? 0 : 1 # Conditionally
use data source
  name                = var.log_analytics_workspace_name # Assumes
name is unique if ID is not directly provided
  resource_group_name = split("/",
var.existing_log_analytics_workspace_id)[4] # Extract RG name from ID
}

# Application Insights
resource "azurerm_application_insights" "main" {
  name                = "appi-${var.environment}-aihub"
  location            = azurerm_resource_group.monitoring_rg.location
  resource_group_name = azurerm_resource_group.monitoring_rg.name
  application_type    = "Web" # Or "Other" or "Node.JS" etc.
  workspace_id        = var.create_log_analytics_workspace ?
azurerm_log_analytics_workspace.main[0].id :
data.azurerm_log_analytics_workspace.existing[0].id

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

output "log_analytics_workspace_id" {
  value = var.create_log_analytics_workspace ?
azurerm_log_analytics_workspace.main[0].id :
data.azurerm_log_analytics_workspace.existing[0].id
  description = "The ID of the Log Analytics Workspace."
}

output "application_insights_id" {
  value = azurerm_application_insights.main.id
  description = "The ID of the Application Insights instance."
}

output "application_insights_instrumentation_key" {
  value = azurerm_application_insights.main.instrumentation_key
  description = "The Instrumentation Key for Application Insights."
  sensitive   = true
}
```

modules/monitoring/variables.tf

```
# modules/monitoring/variables.tf
variable "resource_group_name" {
  description = "The name of the resource group for monitoring."
  type        = string
}

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

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

variable "create_log_analytics_workspace" {
  description = "Set to true to create a new Log Analytics Workspace,
false to use an existing one."
  type        = bool
}

variable "existing_log_analytics_workspace_id" {
  description = "ID of an existing Log Analytics Workspace (if
create_new_log_analytics_workspace is false)."
  type        = string
  default     = null
}

variable "log_analytics_workspace_name" {
  description = "Name for the new Log Analytics Workspace (if created)
or the existing one."
  type        = string
}
```

modules/cost_management

This module would include Event Hubs, Function Apps for processing cost data, and potentially Power BI integration (though Power BI configuration is usually outside Terraform for detailed dashboards).

```
# modules/cost_management/main.tf
resource "azurerm_resource_group" "cost_rg" {
  name     = var.resource_group_name
  location = var.location
}

# Event Hub Namespace for cost data ingestion
resource "azurerm_eventhub_namespace" "cost_events_namespace" {
  name                = "evhns-${var.environment}-cost"
  location            = azurerm_resource_group.cost_rg.location
  resource_group_name = azurerm_resource_group.cost_rg.name
  sku                 = "Standard"
  capacity            = 1

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

# Event Hub for cost data
resource "azurerm_eventhub" "cost_event_hub" {
  name                = "cost-data"
  namespace_name      =
azurerm_eventhub_namespace.cost_events_namespace.name
  resource_group_name = azurerm_resource_group.cost_rg.name
  partition_count     = 2
  message_retention_in_days = 1

  # Capture data to storage account (for processing by Function App
later)
  capture_enabled = true
  capture_container_name =
azurerm_storage_container.cost_data_container.name
  capture_encoding       = "Avro"
  capture_interval_in_seconds = 300
  capture_size_limit_in_bytes = 10485760
  capture_destination_id =
azurerm_storage_account.cost_func_storage.id # Use storage account
from shared services
}

# Storage Account for Function App (if not using shared_services)
resource "azurerm_storage_account" "cost_func_storage" {
  name                     = "sacostfunc${var.environment}aihub" #
Must be globally unique
  resource_group_name      = azurerm_resource_group.cost_rg.name
  location                 = azurerm_resource_group.cost_rg.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
}

resource "azurerm_storage_container" "cost_data_container" {
  name                  = "cost-capture"
  storage_account_name  =
azurerm_storage_account.cost_func_storage.name
  container_access_type = "private"
}

# Function App for processing Event Hub data and pushing to Power
BI/other sinks
resource "azurerm_app_service_plan" "cost_func_plan" {
  name                = "asp-costfunc-${var.environment}"
  location            = azurerm_resource_group.cost_rg.location
  resource_group_name = azurerm_resource_group.cost_rg.name
  kind                = "FunctionApp"
  sku {
    tier = "Consumption" # Or "Premium" for dedicated
    size = "Y1"
  }
  tags = {
    environment = var.environment
    project     = "AIHub"
    purpose     = "CostManagement"
  }
}

resource "azurerm_function_app" "cost_processor_func" {
  name                       = "func-costprocessor-${var.environment}"
  location                   = azurerm_resource_group.cost_rg.location
  resource_group_name        = azurerm_resource_group.cost_rg.name
  app_service_plan_id        =
azurerm_app_service_plan.cost_func_plan.id
  storage_account_name       =
azurerm_storage_account.cost_func_storage.name
  storage_account_access_key =
azurerm_storage_account.cost_func_storage.primary_access_key
  version                    = "~4" # .NET 6, Node 16, Python 3.9 etc.
  app_settings = {
    "FUNCTIONS_WORKER_RUNTIME" = "python" # Or "dotnet", "node"
    "EventHubConnection"       =
azurerm_eventhub_namespace.cost_events_namespace.default_primary_conne
ction_string
    "WEBSITES_ENABLE_APP_SERVICE_STORAGE" = "false" # Use Azure Files
for content
    "APPLICATIONINSIGHTS_CONNECTION_STRING" =
var.application_insights_connection_string # From monitoring module
  }

  identity {
    type = "SystemAssigned"
  }

  lifecycle {
    ignore_changes = [
      app_settings["WEBSITE_RUN_FROM_PACKAGE"], # Often managed by
pipeline for deployments
    ]
  }

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

# Example: Assign Function App permissions to read cost data or write
to Power BI
# This depends on how you integrate with Power BI (e.g., Dataflows,
APIs)
resource "azurerm_role_assignment" "func_storage_reader" {
  scope                = azurerm_storage_account.cost_func_storage.id
  role_definition_name = "Storage Blob Data Reader"
  principal_id         =
azurerm_function_app.cost_processor_func.identity[0].principal_id
}

output "event_hub_namespace_id" {
  value = azurerm_eventhub_namespace.cost_events_namespace.id
  description = "The ID of the Event Hub Namespace for cost data."
}

output "cost_function_app_name" {
  value = azurerm_function_app.cost_processor_func.name
  description = "The name of the Function App for cost processing."
}
```

modules/cost_management/variables.tf

```
# modules/cost_management/variables.tf
variable "resource_group_name" {
  description = "The name of the resource group for cost management."
  type        = string
}

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

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

variable "log_analytics_workspace_id" {
  description = "The ID of the Log Analytics Workspace for function
app monitoring."
  type        = string
}

variable "event_hub_namespace_id" {
  description = "The ID of the Event Hub Namespace (if using
shared_services)."
  type        = string
  default     = null # If created within this module
}

variable "storage_account_id" {
  description = "The ID of the Storage Account (if using
shared_services)."
  type        = string
  default     = null # If created within this module
}

variable "application_insights_connection_string" {
  description = "Connection string for Application Insights."
  type        = string
  sensitive   = true
}
```

modules/shared_services

This module can host resources that are shared across other modules, like Key Vault for secrets (e.g., OpenAI API key), or a shared storage account.

```
# modules/shared_services/main.tf
resource "azurerm_resource_group" "shared_rg" {
  name     = var.resource_group_name
  location = var.location
}

# Key Vault
resource "azurerm_key_vault" "main" {
  name                       = "kv-aihub-${var.environment}"
  location                   =
azurerm_resource_group.shared_rg.location
  resource_group_name        = azurerm_resource_group.shared_rg.name
  tenant_id                  =
data.azurerm_client_config.current.tenant_id
  sku_name                   = "standard"
  soft_delete_retention_days = 7 # Adjust as per policy

  access_policy {
    tenant_id = data.azurerm_client_config.current.tenant_id
    object_id = data.azurerm_client_config.current.object_id # Your
pipeline/service principal
    key_permissions = [
      "Get", "List", "Create", "Delete", "Recover", "Backup",
"Restore", "Purge", "Decrypt", "Encrypt", "Sign", "Verify",
"UnwrapKey", "WrapKey"
    ]
    secret_permissions = [
      "Get", "List", "Set", "Delete", "Recover", "Backup", "Restore",
"Purge"
    ]
    certificate_permissions = [
      "Get", "List", "Create", "Import", "Delete", "Recover",
"Backup", "Restore", "ManageContacts", "ManageIssuers", "GetIssuers",
"ListIssuers", "SetIssuers", "DeleteIssuers", "Purge"
    ]
  }

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

# Data source for the current Azure client configuration (for Key
Vault access policy)
data "azurerm_client_config" "current" {}

# Store OpenAI API Key in Key Vault
resource "azurerm_key_vault_secret" "openai_api_key" {
  name         = "OpenAIApiKey"
  value        = var.openai_api_key # Pass as sensitive variable
  key_vault_id = azurerm_key_vault.main.id
  content_type = "text/plain"
}
```

```
# Storage Account for general purposes (e.g., Function App code, AI
Hub data)
resource "azurerm_storage_account" "general_storage" {
  name                     = "sagen${var.environment}aihub" # Must be
globally unique
  resource_group_name      = azurerm_resource_group.shared_rg.name
  location                 = azurerm_resource_group.shared_rg.location
  account_tier             = "Standard"
  account_replication_type = "LRS" # Change to GRS/ZRS for prod

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

output "key_vault_id" {
  value = azurerm_key_vault.main.id
  description = "The ID of the Key Vault."
}

output "storage_account_id" {
  value = azurerm_storage_account.general_storage.id
  description = "The ID of the general purpose Storage Account."
}

output "openai_api_key_secret_id" {
  value = azurerm_key_vault_secret.openai_api_key.id
  description = "The ID of the OpenAI API Key secret in Key Vault."
  sensitive = true
}
```

modules/shared_services/variables.tf

```
# modules/shared_services/variables.tf
variable "resource_group_name" {
  description = "The name of the resource group for shared services."
  type        = string
}

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

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

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

variable "openai_api_key" {
  description = "The API key for Azure OpenAI Service. Store this
securely (e.g., in Azure DevOps secret variable)."
  type        = string
  sensitive   = true
}
```

Azure Pipelines (azure-pipelines.yml)

This YAML file will define your CI/CD pipeline to deploy the Terraform code.

```
# azure-pipelines.yml
trigger:
  branches:
    include:
      - main
      - feature/*
  paths:
    include:
      - terraform/**/* # Trigger on changes in terraform folder

pr:
  branches:
    include:
      - main

variables:
  - group: tf-dev-secrets # Link to Variable Group in Azure DevOps for
Dev secrets
  - group: tf-qa-secrets # Link to Variable Group in Azure DevOps for
QA secrets
  - group: tf-prod-secrets # Link to Variable Group in Azure DevOps
for Prod secrets
  - name: terraformWorkingDirectory
    value: '$(Build.SourcesDirectory)/environments' # Point to the
environments folder

stages:
  - stage: TerraformPlan
    displayName: 'Terraform Plan'
    jobs:
      - job: PlanDev
        displayName: 'Plan Dev Environment'
        pool:
          vmImage: 'ubuntu-latest'
        steps:
          - task: AzureCLI@2
```

```
            displayName: 'Azure Login (Service Principal)'
            inputs:
              azureSubscription: 'AzureDevSubscription' # Service
Connection Name
              scriptType: 'bash'
              scriptLocation: 'inlineScript'
              inlineScript: |
                az account show

          - task: TerraformInstaller@1
            displayName: 'Install Terraform'
            inputs:
              terraformVersion: 'latest'

          - task: CmdLine@2
            displayName: 'Initialize Terraform (Dev)'
            inputs:
              script: |
                terraform init -backend-
config="resource_group_name=$(tfstateDevResourceGroup)" -backend-
config="storage_account_name=$(tfstateDevStorageAccount)" -backend-
config="container_name=tfstate" -backend-config="key=aihub-
accelerator-dev.tfstate" -backend-
config="subscription_id=$(DevSubscriptionId)"
              workingDirectory: '$(terraformWorkingDirectory)/dev'

          - task: CmdLine@2
            displayName: 'Terraform Plan (Dev)'
            inputs:
              script: |
                terraform plan -out=tfplan_dev.binary -
var="environment=dev" -
var="existing_spoke_resource_group_name=$(existingSpokeResourceGroupDe
v)" -var="existing_vnet_name=$(existingVnetNameDev)" -
var="existing_apim_subnet_name=$(existingApimSubnetNameDev)" -
var="existing_aihub_subnet_name=$(existingAihubSubnetNameDev)" -
var="openai_api_key=$(OpenAIApiKeyDev)" -
var="application_insights_connection_string=$(AppInsightsConnectionStr
ingDev)"
              workingDirectory: '$(terraformWorkingDirectory)/dev'

          - publish:
$(terraformWorkingDirectory)/dev/tfplan_dev.binary
            artifact: tfplan_dev

      - job: PlanQA
        displayName: 'Plan QA Environment'
        # ... similar steps as PlanDev, but target QA variables and
service connection
        # ... and `terraform init -backend-config` for QA, and
`terraform plan` for QA.
        # Ensure to link `tf-qa-secrets` variable group.
        pool:
          vmImage: 'ubuntu-latest'
        steps:
          - task: AzureCLI@2
            displayName: 'Azure Login (Service Principal)'
            inputs:
              azureSubscription: 'AzureQASubscription' # Service
Connection Name
              scriptType: 'bash'
              scriptLocation: 'inlineScript'
              inlineScript: |
                az account show

          - task: TerraformInstaller@1
            displayName: 'Install Terraform'
            inputs:
              terraformVersion: 'latest'

          - task: CmdLine@2
            displayName: 'Initialize Terraform (QA)'
            inputs:
              script: |
                terraform init -backend-
config="resource_group_name=$(tfstateQAResourceGroup)" -backend-
config="storage_account_name=$(tfstateQAStorageAccount)" -backend-
config="container_name=tfstate" -backend-config="key=aihub-
accelerator-qa.tfstate" -backend-
config="subscription_id=$(QASubscriptionId)"
              workingDirectory: '$(terraformWorkingDirectory)/qa'

          - task: CmdLine@2
            displayName: 'Terraform Plan (QA)'
            inputs:
              script: |
                terraform plan -out=tfplan_qa.binary -
var="environment=qa" -
var="existing_spoke_resource_group_name=$(existingSpokeResourceGroupQA
)" -var="existing_vnet_name=$(existingVnetNameQA)" -
var="existing_apim_subnet_name=$(existingApimSubnetNameQA)" -
var="existing_aihub_subnet_name=$(existingAihubSubnetNameQA)" -
var="openai_api_key=$(OpenAIApiKeyQA)" -
var="application_insights_connection_string=$(AppInsightsConnectionStr
ingQA)"
              workingDirectory: '$(terraformWorkingDirectory)/qa'

          - publish: $(terraformWorkingDirectory)/qa/tfplan_qa.binary
            artifact: tfplan_qa


  - stage: TerraformApply
    displayName: 'Terraform Apply'
    dependsOn: TerraformPlan
    condition: succeeded()
    jobs:
      - deployment: DeployDev
        displayName: 'Deploy Dev Environment'
        environment: 'dev' # Link to Azure DevOps Environment for
approvals
        pool:
          vmImage: 'ubuntu-latest'
        strategy:
          runOnce:
            preDeploy:
              steps:
                - task: DownloadPipelineArtifact@2
                  inputs:
                    artifact: tfplan_dev
                    path: $(terraformWorkingDirectory)/dev
                - task: AzureCLI@2
                  displayName: 'Azure Login (Service Principal)'
                  inputs:
                    azureSubscription: 'AzureDevSubscription'
                    scriptType: 'bash'
                    scriptLocation: 'inlineScript'
                    inlineScript: |
                      az account show
                - task: TerraformInstaller@1
                  displayName: 'Install Terraform'
                  inputs:
                    terraformVersion: 'latest'

            deploy:
              steps:
                - task: CmdLine@2
                  displayName: 'Initialize Terraform (Dev - Apply)'
                  inputs:
                    script: |
                      terraform init -backend-
config="resource_group_name=$(tfstateDevResourceGroup)" -backend-
config="storage_account_name=$(tfstateDevStorageAccount)" -backend-
config="container_name=tfstate" -backend-config="key=aihub-
accelerator-dev.tfstate" -backend-
config="subscription_id=$(DevSubscriptionId)"
                    workingDirectory:
'$(terraformWorkingDirectory)/dev'

                - task: CmdLine@2
                  displayName: 'Terraform Apply (Dev)'
                  inputs:
                    script: |
                      terraform apply tfplan_dev.binary
                    workingDirectory:
'$(terraformWorkingDirectory)/dev'
```

```
      - deployment: DeployQA
        displayName: 'Deploy QA Environment'
        environment: 'qa' # Link to Azure DevOps Environment for
approvals
        # ... similar steps as DeployDev, but target QA artifacts and
service connection
        # ... with a manual approval gate before it runs.
        pool:
          vmImage: 'ubuntu-latest'
        strategy:
          runOnce:
            preDeploy:
              steps:
                - task: DownloadPipelineArtifact@2
                  inputs:
                    artifact: tfplan_qa
                    path: $(terraformWorkingDirectory)/qa
                - task: AzureCLI@2
                  displayName: 'Azure Login (Service Principal)'
                  inputs:
                    azureSubscription: 'AzureQASubscription'
                    scriptType: 'bash'
                    scriptLocation: 'inlineScript'
                    inlineScript: |
                      az account show
                - task: TerraformInstaller@1
                  displayName: 'Install Terraform'
                  inputs:
                    terraformVersion: 'latest'

            deploy:
              steps:
                - task: CmdLine@2
                  displayName: 'Initialize Terraform (QA - Apply)'
                  inputs:
                    script: |
                      terraform init -backend-
config="resource_group_name=$(tfstateQAResourceGroup)" -backend-
config="storage_account_name=$(tfstateQAStorageAccount)" -backend-
config="container_name=tfstate" -backend-config="key=aihub-
accelerator-qa.tfstate" -backend-
config="subscription_id=$(QASubscriptionId)"
                    workingDirectory:
'$(terraformWorkingDirectory)/qa'

                - task: CmdLine@2
                  displayName: 'Terraform Apply (QA)'
                  inputs:
                    script: |
                      terraform apply tfplan_qa.binary
                    workingDirectory:
'$(terraformWorkingDirectory)/qa'
```

```
      - deployment: DeployProd
        displayName: 'Deploy Prod Environment'
        environment: 'prod' # Link to Azure DevOps Environment for
approvals
        # Add even more stringent approval gates for Prod
        # ... similar steps as DeployQA, but target Prod artifacts and
service connection
        # ... and link `tf-prod-secrets` variable group.
        pool:
          vmImage: 'ubuntu-latest'
        strategy:
          runOnce:
            preDeploy:
              steps:
                - task: DownloadPipelineArtifact@2
                  inputs:
                    artifact: tfplan_prod
                    path: $(terraformWorkingDirectory)/prod
                - task: AzureCLI@2
                  displayName: 'Azure Login (Service Principal)'
                  inputs:
                    azureSubscription: 'AzureProdSubscription'
                    scriptType: 'bash'
                    scriptLocation: 'inlineScript'
                    inlineScript: |
                      az account show
                - task: TerraformInstaller@1
                  displayName: 'Install Terraform'
                  inputs:
                    terraformVersion: 'latest'

            deploy:
              steps:
                - task: CmdLine@2
                  displayName: 'Initialize Terraform (Prod - Apply)'
                  inputs:
                    script: |
                      terraform init -backend-
config="resource_group_name=$(tfstateProdResourceGroup)" -backend-
config="storage_account_name=$(tfstateProdStorageAccount)" -backend-
config="container_name=tfstate" -backend-config="key=aihub-
accelerator-prod.tfstate" -backend-
config="subscription_id=$(ProdSubscriptionId)"
                    workingDirectory:
'$(terraformWorkingDirectory)/prod'

                - task: CmdLine@2
                  displayName: 'Terraform Apply (Prod)'
                  inputs:
                    script: |
                      terraform apply tfplan_prod.binary
                    workingDirectory:
'$(terraformWorkingDirectory)/prod'
```

Before you run the pipeline:

1. **Azure DevOps Setup:**
   - Create an Azure DevOps project.
   - Set up **Service Connections** to your Azure subscriptions (Dev, QA, Prod) using Service Principals. Grant these Service Principals appropriate permissions (Contributor role at the subscription level or more granular if preferred).
   - Create **Variable Groups** (e.g., tf-dev-secrets, tf-qa-secrets, tf-prod-secrets) to store sensitive information and environment-specific values like:
     - tfstateDevResourceGroup, tfstateDevStorageAccount, DevSubscriptionId (and similar for QA/Prod)
     - existingSpokeResourceGroupDev, existingVnetNameDev, existingApimSubnetNameDev, existingAihubSubnetNameDev (and similar for QA/Prod)
     - OpenAIApiKeyDev (mark as secret)
     - AppInsightsConnectionStringDev (mark as secret)
   - Create **Environments** in Azure DevOps (e.g., dev, qa, prod) and configure **approvals** for each. This ensures manual gates before deployments to critical environments.
   - **Manually create the Terraform state storage accounts and resource groups** for each environment. Terraform init needs these to exist before it can use them as a backend.
     - az group create --name tfstate-rg-dev --location "East US"
     - az storage account create --name tfstatedevaihub --resource-group tfstate-rg-dev --location "East US" --sku Standard_LRS
     - az storage container create --name tfstate --account-name tfstatedevaihub
     - Repeat for QA and Prod.
2. **Terraform Configuration:**
   - **Review environments/*/variables.tf:** Update the default values for existing resource names (spoke RG, VNet, subnets) to match your actual environment.
   - **Review modules/apim/main.tf:** Update publisher_name, publisher_email, and hostname if you have custom domains. Adjust sku_name based on environment requirements.
   - **Review modules/aihub/main.tf:** Update the Azure OpenAI url for the backend configuration.
   - **Populate modules/apim/policies/:** Add your actual XML policies for global, product, and API levels.
   - **Security:** Ensure the OpenAIApiKey is passed as a *secret variable* in Azure DevOps and never hardcoded in Terraform files. Similarly for AppInsightsConnectionString.
3. **Initial Terraform State Management:**
   - For the *very first* run, you might want to run terraform init and terraform apply locally for one environment (e.g., dev) to ensure the backend is correctly configured and the initial resources are created. Once stable, rely on the pipeline.

Self-Service Pipeline for Application Teams (Separate from this IaC)

As you mentioned, this is separate, but here's a conceptual outline for the self-service part:

- **Trigger:** Application team requests access, perhaps through a web portal, a service catalog tool (like ServiceNow), or by submitting a pull request to a specific repository.
- **Approval Workflow:** The request triggers an Azure DevOps Release pipeline (or GitHub Actions workflow). This pipeline includes a manual approval gate where your team reviews the request.
- **Automated Onboarding (after approval):**
  - **Dynamic API Creation in APIM:**
    - The self-service pipeline could use the Azure CLI or PowerShell to call the APIM management API directly.
    - It would create new APIM APIs, potentially import OpenAPI specs for the application's APIs, and link them to existing products.
    - It could also assign custom policies specific to the application.
  - **Azure AD App Registration/Service Principal:** Create an Azure AD application registration and service principal for the application, granting it necessary permissions to interact with AI Hub services.
  - **Key Vault Secrets:** Store application-specific secrets (e.g., API keys for the new APIM API, connection strings) in a dedicated Key Vault.
  - **Monitoring Integration:** Automatically configure logging for the new application's APIM usage into the existing Log Analytics Workspace.
  - **Output/Notification:** Provide the application team with the necessary API keys, endpoint URLs, and documentation via email, a ticketing system, or a dedicated portal.

This self-service pipeline would *consume* the infrastructure provisioned by your core Terraform IaC (the one we just designed). It would not manage the core APIM service, AI Hub, or monitoring infrastructure itself, but rather *configure* and *onboard* applications onto that existing infrastructure.

This comprehensive setup provides the flexibility, modularity, and control you need for your Azure AI Hub Landing Zone accelerator, ensuring consistency and manageability across all your environments.