

This is an excellent and comprehensive request! Setting up a robust Azure AI Hub Landing Zone with Terraform is a critical step for scalable AI initiatives. Let's break this down into a well-organized and structured Terraform project.

The core idea is to use a modular approach, allowing for flexibility in resource provisioning and environment-specific configurations.

Overall Structure and Design Principles

1. **Modular Design:** Each major component (APIM, Monitoring, Cost Management, AI Hub itself) will be a separate Terraform module. This promotes reusability, maintainability, and allows for conditional inclusion/exclusion.
2. **Environment-Specific Configuration:** We'll use .tfvars files for different environments (e.g., dev.tfvars, test.tfvars, prod.tfvars) to manage environment-specific values.
3. **Conditional Resource Creation:** Leverage Terraform's count and for_each meta-arguments, along with input variables, to conditionally create or refer to existing resources.
4. **Data Sources for Existing Resources:** Utilize data blocks to reference pre-existing resources like VNets, Subnets, Subscriptions, and Resource Groups.
5. **Azure DevOps Pipelines:** Outline the structure for Azure Pipelines to deploy these Terraform configurations.
6. **Self-Service Onboarding (Separate):** As requested, the application onboarding will be a separate pipeline, interacting with the pre-provisioned AI Hub infrastructure.

Project Structure (File System)

```
.
├── environments/
│   ├── dev.tfvars
│   ├── test.tfvars
│   └── prod.tfvars
├── modules/
│   ├── aihub/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── apim/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── monitoring/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   ├── cost_management/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── outputs.tf
│   └── network/
```

```

|         |— main.tf
|         |— variables.tf
|         |— outputs.tf
|— main.tf
|— variables.tf
|— versions.tf
|— README.md
|— azure-pipelines.yml

```

Core Terraform Files

versions.tf (Root)

```

terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = "~> 3.0" # Use a suitable version
    }
  }
  required_version = ">= 1.0"
}

provider "azurerm" {
  features {}
}

```

variables.tf (Root)

This file will define the global variables that can be overridden by environment-specific .tfvars files.

Global Variables

```

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

variable "location" {
  description = "The Azure region where resources will be deployed."
  type        = string
}

variable "resource_group_name_prefix" {
  description = "Prefix for resource group names. A common suffix will

```

```

be added."
    type      = string
    default   = "rg"
}

variable "tags" {
    description = "A map of tags to apply to all resources."
    type        = map(string)
    default     = {}
}

# Existing Network Details
variable "existing_vnet_name" {
    description = "Name of the existing VNet."
    type        = string
}

variable "existing_vnet_resource_group_name" {
    description = "Resource Group name of the existing VNet."
    type        = string
}

variable "existing_apim_subnet_name" {
    description = "Name of the existing subnet for APIM."
    type        = string
}

variable "existing_aihub_subnet_name" {
    description = "Name of the existing subnet for AI Hub private
endpoints."
    type        = string
}

# Feature Flags for conditional deployment/inclusion
variable "enable_apim" {
    description = "Whether to deploy Azure API Management."
    type        = bool
    default     = true
}

variable "enable_cost_management" {
    description = "Whether to deploy cost management resources."
    type        = bool
    default     = true
}

variable "enable_monitoring" {
    description = "Whether to deploy monitoring resources (Log

```

```

Analytics, App Insights, Event Hub, Functions)."
    type          = bool
    default       = true
}

# Variables for injecting existing resources
variable "use_existing_log_analytics_workspace" {
    description = "Set to true to use an existing Log Analytics
Workspace."
    type        = bool
    default     = false
}

variable "existing_log_analytics_workspace_id" {
    description = "The resource ID of the existing Log Analytics
Workspace to use."
    type        = string
    default     = null
}

variable "use_existing_app_insights" {
    description = "Set to true to use an existing Application Insights
resource."
    type        = bool
    default     = false
}

variable "existing_app_insights_id" {
    description = "The resource ID of the existing Application Insights
resource to use."
    type        = string
    default     = null
}

# APIM specific variables
variable "apim_sku_name" {
    description = "SKU for API Management (e.g., 'Developer_1',
'Standard_V2')."
    type        = string
    default     = "Developer_1" # Use Standard_V2 or Premium for
production
}

variable "apim_publisher_email" {
    description = "Publisher email for API Management."
    type        = string
}

```

```

variable "apim_publisher_name" {
  description = "Publisher name for API Management."
  type        = string
}

# AI Hub specific variables
variable "aihub_name" {
  description = "Name for the Azure AI Hub."
  type        = string
}

# Cost Management specific variables
variable "cost_management_budget_amount" {
  description = "The amount for the budget alert (e.g., 1000 for $1000)."
  type        = number
  default     = 500
}

variable "cost_management_budget_time_grain" {
  description = "The time grain for the budget (e.g., 'Monthly', 'Quarterly', 'Annually')."
  type        = string
  default     = "Monthly"
}

variable "cost_management_budget_notification_emails" {
  description = "A list of email addresses to receive budget alerts."
  type        = list(string)
  default     = []
}

# Monitoring specific variables
variable "event_hub_namespace_sku" {
  description = "SKU for Event Hub Namespace (e.g., 'Standard', 'Basic')."
  type        = string
  default     = "Standard"
}

variable "event_hub_name" {
  description = "Name of the Event Hub."
  type        = string
  default     = "aihublogs"
}

variable "function_app_name" {
  description = "Name for the Azure Function App."

```

```

    type          = string
}

variable "function_app_runtime" {
  description = "Runtime for the Function App (e.g., 'node', 'dotnet',
'python')."
  type        = string
  default     = "dotnet" # Example for .NET 8 Isolated
}

variable "power_bi_workspace_id" {
  description = "The ID of the Power BI workspace for integration
(optional)."
  type        = string
  default     = null
}

variable "power_bi_dataset_name" {
  description = "The name of the Power BI dataset for integration
(optional)."
  type        = string
  default     = null
}

```

main.tf (Root)

This file orchestrates the deployment by calling modules and setting up global resources.

```

resource "azurerm_resource_group" "aihub_rg" {
  name     =
"${var.resource_group_name_prefix}-${var.environment}-aihub"
  location = var.location
  tags     = var.tags
}

# Data sources for existing network components
data "azurerm_virtual_network" "existing_vnet" {
  name                     = var.existing_vnet_name
  resource_group_name     = var.existing_vnet_resource_group_name
}

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

```

```

data "azurerm_subnet" "aihub_private_endpoint_subnet" {
  name = var.existing_aihub_subnet_name
  virtual_network_name =
data.azurerm_virtual_network.existing_vnet.name
  resource_group_name =
data.azurerm_virtual_network.existing_vnet.resource_group_name
}

#
# Module Calls
#

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

  name = var.aihub_name
  resource_group_name = azurerm_resource_group.aihub_rg.name
  location = var.location
  tags = var.tags

  # Private endpoint integration
  vnet_id =
data.azurerm_virtual_network.existing_vnet.id
  private_endpoint_subnet_id =
data.azurerm_subnet.aihub_private_endpoint_subnet.id
}

# API Management Module
module "apim" {
  count = var.enable_apim ? 1 : 0

  source = "../modules/apim"

  name = "${var.aihub_name}-apim-${var.environment}"
  resource_group_name = azurerm_resource_group.aihub_rg.name
  location = var.location
  sku_name = var.apim_sku_name
  publisher_email = var.apim_publisher_email
  publisher_name = var.apim_publisher_name
  tags = var.tags

  # VNet integration for Internal/External mode
  vnet_type = "External" # Or "Internal" if you want full private
access
  subnet_id = data.azurerm_subnet.apim_subnet.id

```

```

    # Pass AI Hub endpoint for API configuration
    aihub_endpoint = module.aihub.aihub_endpoint
}

# Monitoring Module
module "monitoring" {
    count = var.enable_monitoring ? 1 : 0

    source = "./modules/monitoring"

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

    use_existing_log_analytics_workspace =
var.use_existing_log_analytics_workspace
    existing_log_analytics_workspace_id =
var.existing_log_analytics_workspace_id

    use_existing_app_insights = var.use_existing_app_insights
    existing_app_insights_id = var.existing_app_insights_id

    event_hub_namespace_sku      = var.event_hub_namespace_sku
    event_hub_name                = var.event_hub_name
    function_app_name             = var.function_app_name
    function_app_runtime          = var.function_app_runtime
    power_bi_workspace_id         = var.power_bi_workspace_id
    power_bi_dataset_name         = var.power_bi_dataset_name

    # Optionally, pass APIM ID if you want to configure APIM diagnostics
    to send to Log Analytics
    apim_id = try(module.apim[0].apim_id, null)
}

# Cost Management Module
module "cost_management" {
    count = var.enable_cost_management ? 1 : 0

    source = "./modules/cost_management"

    resource_group_name = azurerm_resource_group.aihub_rg.name
    subscription_id      = data.azurerm_subscription.current.id #
Requires data.azurerm_subscription
    budget_amount         = var.cost_management_budget_amount
    time_grain            = var.cost_management_budget_time_grain
    notification_emails   = var.cost_management_budget_notification_emails
    tags                  = var.tags

```



```
}
```

```
# Data source for current subscription (used by cost management)
data "azurerm_subscription" "current" {}
```

Module Implementations

modules/aihub/main.tf

```
resource "azurerm_ai_hub" "main" {
  name                        = var.name
  resource_group_name       = var.resource_group_name
  location                  = var.location
  tags                      = var.tags

  # Example of integrating with existing Cognitive Services if desired
  # cognitive_services_id =
  "/subscriptions/<sub-id>/resourceGroups/<rg-name>/providers/Microsoft.
CognitiveServices/accounts/<account-name>"

  # Network isolation with Private Endpoint
  network_isolation_mode = "Disabled" # Or "Enabled" if you want full
isolation
  # if "Enabled", you would also need to configure private endpoints
and DNS
  # private_endpoint_connection {
  #   name = "${var.name}-pe"
  #   subnet_id = var.private_endpoint_subnet_id
  #   is_manual_connection = false
  #   request_message = "Automated Private Endpoint connection"
  # }
}

# Example of Private Endpoint for AI Hub (if network_isolation_mode is
"Enabled")
resource "azurerm_private_endpoint" "aihub_pe" {
  # count = var.network_isolation_mode == "Enabled" ? 1 : 0 #
Uncomment if you enable network isolation for AI Hub
  count = 0 # Currently disabled for simplicity, enable if AI Hub
requires private access

  name                        = "${var.name}-pe"
  location                  = var.location
  resource_group_name       = var.resource_group_name
  subnet_id                 = var.private_endpoint_subnet_id
}
```

```

    private_service_connection {
      name = "${var.name}-psc"
      is_manual_connection = false
      private_connection_resource_id = azurerm_ai_hub.main.id
      subresource_names = ["aihub"] # Adjust subresource
name if needed
    }

    private_dns_zone_group {
      name = "default"
      private_dns_zone_ids =
[azurerm_private_dns_zone.aihub_private_dns_zone[0].id] # Needs the
DNS Zone
    }
  }

# Private DNS Zone for AI Hub (if network_isolation_mode is "Enabled"
and private endpoint is used)
resource "azurerm_private_dns_zone" "aihub_private_dns_zone" {
  # count = var.network_isolation_mode == "Enabled" ? 1 : 0
  count = 0

  name = "privatelink.api.azureml.ms" # Common DNS zone
for Azure ML/AI Hub
  resource_group_name = var.resource_group_name
}

resource "azurerm_private_dns_zone_virtual_network_link"
"aihub_dns_link" {
  # count = var.network_isolation_mode == "Enabled" ? 1 : 0
  count = 0

  name = "${var.name}-dns-link"
  resource_group_name = var.resource_group_name
  private_dns_zone_name =
azurerm_private_dns_zone.aihub_private_dns_zone[0].name
  virtual_network_id = var.vnet_id
}

```

modules/aihub/variables.tf

```

variable "name" {
  description = "The name of the Azure AI Hub."
  type = string
}

variable "resource_group_name" {

```

```

    description = "The name of the resource group where the AI Hub will
be deployed."
    type        = string
}

variable "location" {
    description = "The Azure region where the AI Hub will be deployed."
    type        = string
}

variable "tags" {
    description = "A map of tags to apply to the AI Hub resource."
    type        = map(string)
    default     = {}
}

# Private Endpoint variables
variable "vnet_id" {
    description = "The ID of the existing Virtual Network for private
endpoint."
    type        = string
}

variable "private_endpoint_subnet_id" {
    description = "The ID of the existing subnet for the AI Hub private
endpoint."
    type        = string
}

```

modules/aihub/outputs.tf

```

output "aihub_id" {
    description = "The ID of the Azure AI Hub."
    value       = azurerm_ai_hub.main.id
}

output "aihub_endpoint" {
    description = "The primary endpoint of the Azure AI Hub."
    value       = azurerm_ai_hub.main.primary_hub_uri # Adjust if this
changes
}

```

modules/apim/main.tf

```

resource "azurerm_api_management_service" "main" {
    name = var.name
}

```

```

location                = var.location
resource_group_name     = var.resource_group_name
publisher_name          = var.publisher_name
publisher_email         = var.publisher_email
sku_name                = var.sku_name
tags                    = var.tags

# VNet integration (Internal/External mode)
# Ensure your APIM SKU supports VNet integration (Developer,
Standard_V2, Premium)
virtual_network_configuration {
    subnet_id = var.subnet_id
}
virtual_network_type = var.vnet_type # "External" or "Internal"
}

# Example API for Azure OpenAI Model
# This demonstrates how to onboard an OpenAI model.
# You would need to retrieve the actual OpenAI service endpoint and
API key.
resource "azurerm_api_management_api" "openai_proxy_api" {
    api_management_id = azurerm_api_management_service.main.id
    display_name      = "OpenAI Model Proxy"
    path              = "openai" # e.g.,
/openai/deployments/gpt-35-turbo/chat/completions
    protocols         = ["https"]
    service_url       = var.aihub_endpoint # This will be the AI Hub's
endpoint
    # You'd typically point this to the *actual* Azure OpenAI endpoint
    # For AI Hub, it might act as a proxy or orchestrator.
    # Let's assume for now, it's proxying to a generic endpoint or a
specific OpenAI service linked to AI Hub.
    # For true Azure OpenAI, the service_url would be like:
"https://<openai-resource-name>.openai.azure.com"

    revision          = "1"
    api_type          = "http"
    description       = "API to proxy requests to Azure OpenAI models
managed by AI Hub."

    # Placeholder for custom policies
    xml_content = <<XML
<policies>
  <inbound>
    <base />
    <set-header name="api-key" exists-action="override">
      <value>{{api-key}}</value> </set-header>
    </inbound>

```

```

        <backend>
            <base />
        </backend>
        <outbound>
            <base />
        </outbound>
        <on-error>
            <base />
        </on-error>
    </policies>
XML
}

# Example of a Product
resource "azurerm_api_management_product" "openai_product" {
    api_management_name = azurerm_api_management_service.main.name
    resource_group_name =
azurerm_api_management_service.main.resource_group_name
    product_id          = "openai-product"
    display_name        = "OpenAI Access Product"
    subscription_required = true
    approval_required   = true # Requires approval for subscriptions
    published            = true

    # Policies for the product can also be defined here
    # xml_content = <<XML
    # <policies>
    #     <inbound>
    #         <rate-limit-by-key calls="100" renewal-period="60"
counter-key=@"(context.Subscription.Id)" />
    #     </inbound>
    #     <backend>
    #         <base />
    #     </backend>
    #     <outbound>
    #         <base />
    #     </outbound>
    # </policies>
    # XML
}

resource "azurerm_api_management_product_api" "link_openai_to_product"
{
    api_management_name = azurerm_api_management_service.main.name
    resource_group_name =
azurerm_api_management_service.main.resource_group_name
    product_id          =
azurerm_api_management_product.openai_product.product_id

```

```

    api_name =
azurerm_api_management_api.openai_proxy_api.name
}

# Named Value for OpenAI API Key (securely fetch from Key Vault or
similar)
resource "azurerm_api_management_named_value" "openai_api_key" {
  api_management_name = azurerm_api_management_service.main.name
  resource_group_name =
azurerm_api_management_service.main.resource_group_name
  name = "openai-api-key"
  display_name = "OpenAI API Key"
  value = "dummy-api-key-replace-with-keyvault" #
IMPORTANT: Replace with actual key vault reference or secure injection
  # Example for Key Vault reference:
  # value_from_key_vault {
  #   secret_id = "<your-key-vault-secret-id>"
  #   identity_client_id =
azurerm_user_assigned_identity.apim_identity.client_id # If using UAI
  # }
  secret = true
}

# Example: User Assigned Identity for APIM (for Key Vault access)
resource "azurerm_user_assigned_identity" "apim_identity" {
  name = "${var.name}-uai"
  resource_group_name = var.resource_group_name
  location = var.location
  tags = var.tags
}

resource "azurerm_role_assignment" "apim_key_vault_reader" {
  # count = <condition_to_enable_key_vault_access> ? 1 : 0
  scope = azurerm_key_vault.kv_for_apim.id # Replace
with your Key Vault ID
  role_definition_name = "Key Vault Secrets User" # Or "Key Vault
Reader"
  principal_id =
azurerm_user_assigned_identity.apim_identity.principal_id
}

# Example: Key Vault (if APIM needs to fetch secrets)
resource "azurerm_key_vault" "kv_for_apim" {
  name = "${var.name}-kv"
  location = var.location
  resource_group_name = var.resource_group_name
  sku_name = "standard"
  tenant_id =

```

```
data.azure_rm_client_config.current.tenant_id
  soft_delete_retention_days = 7
  purge_protection_enabled   = false
  tags                       = var.tags
}

data "azurerm_client_config" "current" {}
```

modules/apim/variables.tf

```
variable "name" {
  description = "The name of the API Management service."
  type        = string
}

variable "resource_group_name" {
  description = "The name of the resource group where APIM will be
  deployed."
  type        = string
}

variable "location" {
  description = "The Azure region where APIM will be deployed."
  type        = string
}

variable "sku_name" {
  description = "SKU for API Management."
  type        = string
}

variable "publisher_email" {
  description = "Publisher email for API Management."
  type        = string
}

variable "publisher_name" {
  description = "Publisher name for API Management."
  type        = string
}

variable "tags" {
  description = "A map of tags to apply to the APIM resource."
  type        = map(string)
  default     = {}
}
```

```

variable "vnet_type" {
  description = "The type of VNet integration for APIM ('External' or 'Internal')."
  type        = string
}

variable "subnet_id" {
  description = "The ID of the subnet for APIM VNet integration."
  type        = string
}

variable "aihub_endpoint" {
  description = "The endpoint of the Azure AI Hub to be used as a backend for APIM."
  type        = string
}

```

modules/apim/outputs.tf

```

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

output "apim_gateway_url" {
  description = "The URL of the API Management gateway."
  value       = azurerm_api_management_service.main.gateway_url
}

```

modules/monitoring/main.tf

```

# Conditional Log Analytics Workspace
resource "azurerm_log_analytics_workspace" "main" {
  count = var.use_existing_log_analytics_workspace ? 0 : 1

  name           =
  "${var.environment}-log-analytics-${var.resource_group_name}"
  location       = var.location
  resource_group_name = var.resource_group_name
  sku            = "PerGB2018" # Or other suitable SKU
  retention_in_days = 30
  tags           = var.tags
}

data "azurerm_log_analytics_workspace" "existing" {
  count = var.use_existing_log_analytics_workspace ? 1 : 0
}

```



```

    # Assuming existing_log_analytics_workspace_id is provided as a full
resource ID
    # If you only have name and RG, you would need to adjust this
    id = var.existing_log_analytics_workspace_id
}

locals {
    log_analytics_workspace_id =
var.use_existing_log_analytics_workspace ?
data.azurerm_log_analytics_workspace.existing[0].id :
azurerm_log_analytics_workspace.main[0].id
}

# Conditional Application Insights
resource "azurerm_application_insights" "main" {
    count = var.use_existing_app_insights ? 0 : 1

    name                =
"${var.environment}-appinsights-${var.resource_group_name}"
    location            = var.location
    resource_group_name = var.resource_group_name
    application_type    = "web" # Or other suitable type
    workspace_id        = local.log_analytics_workspace_id
    tags               = var.tags
}

data "azurerm_application_insights" "existing" {
    count = var.use_existing_app_insights ? 1 : 0
    id    = var.existing_app_insights_id
}

# Event Hub for logs/events
resource "azurerm_eventhub_namespace" "main" {
    name                = "${var.environment}-ehn"
    location            = var.location
    resource_group_name = var.resource_group_name
    sku                 = var.event_hub_namespace_sku
    capacity            = 1 # Adjust capacity as needed
    tags               = var.tags
}

resource "azurerm_eventhub" "aihub_logs" {
    name                = var.event_hub_name
    namespace_name      = azurerm_eventhub_namespace.main.name
    resource_group_name =
azurerm_eventhub_namespace.main.resource_group_name
    partition_count     = 2 # Adjust partition count
    message_retention_in_days = 1
}

```

```

}

resource "azurerm_eventhub_namespace_authorization_rule" "send_rule" {
  name                = "send"
  namespace_name      = azurerm_eventhub_namespace.main.name
  resource_group_name = azurerm_eventhub_namespace.main.resource_group_name
  listen              = false
  send                = true
  manage              = false
}

# Azure Function for processing Event Hub events
resource "azurerm_storage_account" "function_app_storage" {
  name                = "${lower(replace(var.function_app_name,
  "-", ""))}" # Function App storage names must be lowercase and no
hyphens
  resource_group_name = var.resource_group_name
  location            = var.location
  account_tier        = "Standard"
  account_replication_type = "LRS"
  tags                = var.tags
}

resource "azurerm_application_insights" "function_app_insights" {
  name                = "${var.function_app_name}-appinsights"
  location            = var.location
  resource_group_name = var.resource_group_name
  application_type    = "other" # Or "web"
  workspace_id        = local.log_analytics_workspace_id
  tags                = var.tags
}

resource "azurerm_app_service_plan" "function_app_plan" {
  name                = "${var.function_app_name}-plan"
  location            = var.location
  resource_group_name = var.resource_group_name
  kind                = "FunctionApp"
  sku {
    tier = "Consumption" # Or "PremiumV2" for production
    size = "Y1"
  }
  tags = var.tags
}

resource "azurerm_function_app" "main" {
  name                = var.function_app_name
  location            = var.location

```

```

    resource_group_name      = var.resource_group_name
    app_service_plan_id      =
azurerm_app_service_plan.function_app_plan.id
    storage_account_name     =
azurerm_storage_account.function_app_storage.name
    storage_account_access_key =
azurerm_storage_account.function_app_storage.primary_access_key
    app_insights_key         =
azurerm_application_insights.function_app_insights.instrumentation_key
    app_insights_connection_string =
azurerm_application_insights.function_app_insights.connection_string

    os_type      = "Windows" # Or "Linux"
    https_only   = true

    identity {
        type = "SystemAssigned"
    }

    site_config {
        scm_type = "None"
        application_stack {
            # Example for .NET 8 Isolated
            dotnet_version = "8.0"
            use_dotnet_isolated_runtime = true
        }
    }
}

app_settings = {
    FUNCTIONS_WORKER_RUNTIME = var.function_app_runtime
    EventHubConnectionString =
azurerm_eventhub_namespace.main.default_primary_connection_string # Or
a specific rule
    EventHubName      = azurerm_eventhub.aihub_logs.name
    # Add any other required app settings for your function logic
}
tags = var.tags
}

# Optionally, configure APIM diagnostics to Log Analytics
resource "azurerm_monitor_diagnostic_setting" "apim_diagnostics" {
    count = var.apim_id != null ? 1 : 0

    name                        = "apim-to-loganalytics"
    target_resource_id         = var.apim_id
    log_analytics_workspace_id = local.log_analytics_workspace_id

    metric {

```

```

        category = "AllMetrics"
        enabled   = true
        retention_policy {
            enabled = false
        }
    }

    log {
        category = "GatewayLogs"
        enabled   = true
        retention_policy {
            enabled = false
        }
    }

    log {
        category = "AuditEvent"
        enabled   = true
        retention_policy {
            enabled = false
        }
    }
}

```

Power BI integration - This is more of an Azure Function or Logic App activity

Terraform primarily provisions the underlying resources.

For direct Power BI integration, you might use:

1. Azure Function to push data to Power BI Streaming Datasets (requires Power BI APIs)

2. Azure Data Factory to pull data from Log Analytics/Event Hub and push to Power BI

3. Direct connections from Power BI to Log Analytics Workspace or Azure Data Lake Storage

For simplicity, we'll assume the Function App or another service handles the data flow to Power BI.

modules/monitoring/variables.tf

```

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

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

```

```
    type          = string
}

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

variable "tags" {
    description = "A map of tags to apply to monitoring resources."
    type        = map(string)
    default     = {}
}

variable "use_existing_log_analytics_workspace" {
    description = "Set to true to use an existing Log Analytics
Workspace."
    type        = bool
}

variable "existing_log_analytics_workspace_id" {
    description = "The resource ID of the existing Log Analytics
Workspace to use."
    type        = string
    default     = null
}

variable "use_existing_app_insights" {
    description = "Set to true to use an existing Application Insights
resource."
    type        = bool
}

variable "existing_app_insights_id" {
    description = "The resource ID of the existing Application Insights
resource to use."
    type        = string
    default     = null
}

variable "event_hub_namespace_sku" {
    description = "SKU for Event Hub Namespace."
    type        = string
}

variable "event_hub_name" {
    description = "Name of the Event Hub."
    type        = string
}
```

```

}

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

variable "function_app_runtime" {
  description = "Runtime for the Function App."
  type        = string
}

variable "power_bi_workspace_id" {
  description = "The ID of the Power BI workspace for integration
(optional).\"
  type        = string
  default     = null
}

variable "power_bi_dataset_name" {
  description = "The name of the Power BI dataset for integration
(optional).\"
  type        = string
  default     = null
}

variable "apim_id" {
  description = "The resource ID of the APIM service for diagnostic
settings (optional).\"
  type        = string
  default     = null
}

```

modules/monitoring/outputs.tf

```

output "log_analytics_workspace_id" {
  description = "The ID of the Log Analytics Workspace.\"
  value       = local.log_analytics_workspace_id
}

output "application_insights_id" {
  description = "The ID of the Application Insights resource.\"
  value       = var.use_existing_app_insights ?
data.azurerm_application_insights.existing[0].id :
azurerm_application_insights.main[0].id
}

```

```

output "event_hub_namespace_id" {
  description = "The ID of the Event Hub Namespace."
  value       = azurerm_eventhub_namespace.main.id
}

output "event_hub_id" {
  description = "The ID of the Event Hub."
  value       = azurerm_eventhub.aihub_logs.id
}

output "function_app_id" {
  description = "The ID of the Function App."
  value       = azurerm_function_app.main.id
}

```

modules/cost_management/main.tf

```

resource "azurerm_consumption_budget_subscription" "aihub_budget" {
  name                = "aihub-${var.resource_group_name}-budget"
  subscription_id     = var.subscription_id
  amount              = var.budget_amount
  time_grain          = var.time_grain
  time_period {
    start_date = formatdate("YYYY-MM-01Z", timestamp()) # Budget
    # starts at the beginning of the current month
  }
  notification {
    enabled    = true
    operator   = "GreaterThan"
    threshold  = 90 # % of budget to trigger alert
    contact_emails = var.notification_emails
  }
  tags = var.tags
}

# Example of a Cost Anomaly Alert (if desired)
# This often requires more advanced setup or specific APIs
# resource "azurerm_cost_anomaly_alert" "main" {
#   name                = "${var.resource_group_name}-anomaly-alert"
#   scope               = var.resource_group_id
#   email_addresses     = var.notification_emails
#   tags               = var.tags
# }

```

modules/cost_management/variables.tf

```

variable "resource_group_name" {
  description = "The name of the resource group to apply the budget to
(for naming)."
```

```

  type          = string
}

variable "subscription_id" {
  description = "The ID of the subscription to apply the budget to."
  type        = string
}

variable "budget_amount" {
  description = "The amount for the budget alert."
  type        = number
}

variable "time_grain" {
  description = "The time grain for the budget."
  type        = string
}

variable "notification_emails" {
  description = "A list of email addresses to receive budget alerts."
  type        = list(string)
}

variable "tags" {
  description = "A map of tags to apply to cost management resources."
  type        = map(string)
  default     = {}
}

```

modules/cost_management/outputs.tf

```

output "budget_id" {
  description = "The ID of the Azure Consumption Budget."
  value       =
azurerm_consumption_budget_subscription.aihub_budget.id
}

```

Environment-Specific Variables (environments/*.tfvars)

environments/dev.tfvars

```

environment          = "dev"
location              = "East US"

```



```

apim_sku_name                = "Developer_1"
apim_publisher_email         = "dev-admin@example.com"
apim_publisher_name          = "Dev Team"
aihub_name                   = "dev-aihub"
cost_management_budget_amount = 200
cost_management_budget_notification_emails =
["devops-alerts@example.com"]
function_app_name            = "dev-aihub-func"
existing_vnet_name            = "dev-vnet"
existing_vnet_resource_group_name = "dev-network-rg"
existing_apim_subnet_name     = "apim-subnet"
existing_aihub_subnet_name    = "private-endpoint-subnet"
tags = {
  Environment = "Dev"
  Project     = "AIHub"
}
# Example of using existing resources
use_existing_log_analytics_workspace = false
# existing_log_analytics_workspace_id =
"/subscriptions/xxx/resourceGroups/yyy/providers/Microsoft.Operational
Insights/workspaces/my-existing-la"
enable_cost_management = true
enable_monitoring      = true
enable_apim            = true

```

environments/prod.tfvars

```

environment                = "prod"
location                   = "West Europe"
apim_sku_name              = "Premium" # Use Premium for
production
apim_publisher_email       = "prod-admin@example.com"
apim_publisher_name        = "Operations Team"
aihub_name                 = "prod-aihub"
cost_management_budget_amount = 5000
cost_management_budget_notification_emails =
["prod-alerts@example.com", "finance-team@example.com"]
function_app_name          = "prod-aihub-func"
existing_vnet_name          = "prod-vnet"
existing_vnet_resource_group_name = "prod-network-rg"
existing_apim_subnet_name   = "apim-subnet"
existing_aihub_subnet_name  = "private-endpoint-subnet"
tags = {
  Environment = "Prod"
  Project     = "AIHub"
  Criticality = "High"
}

```

```
# Example of using existing resources in Prod
use_existing_log_analytics_workspace = true
existing_log_analytics_workspace_id =
"/subscriptions/YOUR_PROD_SUB_ID/resourceGroups/PROD_CORE_MONITORING_R
G/providers/Microsoft.OperationalInsights/workspaces/prod-central-log-
analytics"
use_existing_app_insights = true
existing_app_insights_id =
"/subscriptions/YOUR_PROD_SUB_ID/resourceGroups/PROD_CORE_MONITORING_R
G/providers/Microsoft.Insights/components/prod-central-appinsights"
enable_cost_management = true
enable_monitoring = true
enable_apim = true
```

Azure Pipelines (azure-pipelines.yml)

This pipeline will allow you to deploy to different environments.

```
trigger:
  branches:
    include:
      - main
  paths:
    include:
      - terraform/**/* # Trigger on changes in the terraform directory

pr:
  branches:
    include:
      - main
  paths:
    include:
      - terraform/**/*

variables:
  - group: AzureServiceConnection # Link to an Azure DevOps Variable
    Group for service connection details
  - name: terraformWorkingDirectory
    value: '$(Build.SourcesDirectory)/terraform' # Adjust if your
    terraform files are in a subdirectory

stages:
  - stage: TerraformPlan
    displayName: 'Terraform Plan'
    jobs:
      - job: PlanDev
        displayName: 'Plan Dev Environment'
        pool:
```

```

        vmImage: 'ubuntu-latest'
steps:
  - checkout: self
  - task: AzureCLI@2
    displayName: 'Azure Login'
    inputs:
      azureSubscription: $(AZURE_SERVICE_CONNECTION) # Your
service connection name
      scriptType: 'bash'
      scriptLocation: 'inlineScript'
      inlineScript: |
        az account show
  - script: |
    terraform init
-backend-config="resource_group_name=$(TFSTATE_RG) "
-backend-config="storage_account_name=$(TFSTATE_SA) "
-backend-config="container_name=$(TFSTATE_CONTAINER) "
-backend-config="key=$(TFSTATE_KEY)-dev.tfstate"
    terraform plan -var-file=environments/dev.tfvars
-out=tfplan-dev
    workingDirectory: '$(terraformWorkingDirectory)'
    displayName: 'Terraform Init and Plan (Dev)'
  - publish: '$(terraformWorkingDirectory)/tfplan-dev'
    artifact: 'tfplan-dev'
    displayName: 'Publish tfplan-dev artifact'

- job: PlanProd
  displayName: 'Plan Prod Environment'
  pool:
    vmImage: 'ubuntu-latest'
  steps:
    - checkout: self
    - task: AzureCLI@2
      displayName: 'Azure Login'
      inputs:
        azureSubscription: $(AZURE_SERVICE_CONNECTION)
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          az account show
    - script: |
      terraform init
-backend-config="resource_group_name=$(TFSTATE_RG) "
-backend-config="storage_account_name=$(TFSTATE_SA) "
-backend-config="container_name=$(TFSTATE_CONTAINER) "
-backend-config="key=$(TFSTATE_KEY)-prod.tfstate"
      terraform plan -var-file=environments/prod.tfvars
-out=tfplan-prod

```

```

        workingDirectory: '$(terraformWorkingDirectory)'
        displayName: 'Terraform Init and Plan (Prod)'
    - publish: '$(terraformWorkingDirectory)/tfplan-prod'
      artifact: 'tfplan-prod'
      displayName: 'Publish tfplan-prod artifact'

- stage: TerraformApply
  displayName: 'Terraform Apply'
  dependsOn: TerraformPlan
  condition: succeeded()
  jobs:
    - deployment: DeployDev
      displayName: 'Deploy Dev Environment'
      environment: 'Dev' # Azure DevOps Environment for approvals
      pool:
        vmImage: 'ubuntu-latest'
      strategy:
        runOnce:
          preDeploy:
            steps:
              - download: current
                artifact: tfplan-dev
              - task: AzureCLI@2
                displayName: 'Azure Login'
                inputs:
                  azureSubscription: $(AZURE_SERVICE_CONNECTION)
                  scriptType: 'bash'
                  scriptLocation: 'inlineScript'
                  inlineScript: |
                    az account show
          deploy:
            steps:
              - script: |
                  terraform init
-backend-config="resource_group_name=$(TFSTATE_RG) "
-backend-config="storage_account_name=$(TFSTATE_SA) "
-backend-config="container_name=$(TFSTATE_CONTAINER) "
-backend-config="key=$(TFSTATE_KEY)-dev.tfstate"
                  terraform apply tfplan-dev
                  workingDirectory: '$(terraformWorkingDirectory)'
                  displayName: 'Terraform Apply (Dev)'

    - deployment: DeployProd
      displayName: 'Deploy Prod Environment'
      environment: 'Prod' # Azure DevOps Environment for approvals
      pool:
        vmImage: 'ubuntu-latest'
      strategy:

```



```
key, will be suffixed by environment
}
}
```

- **Azure DevOps Variable Group:** Create a variable group in Azure DevOps (e.g., AzureServiceConnection) and define variables for your backend:
 - AZURE_SERVICE_CONNECTION: Name of your Azure RM Service Connection.
 - TFSTATE_RG: Resource group name for Terraform state.
 - TFSTATE_SA: Storage account name for Terraform state.
 - TFSTATE_CONTAINER: Container name for Terraform state.
 - TFSTATE_KEY: Base key for Terraform state file (e.g., aihub-accelerator). The pipeline script will append -dev.tfstate or -prod.tfstate.

3. Permissions:

- The Service Principal used by your Azure DevOps pipeline needs appropriate permissions (e.g., Contributor role) on the target subscription and resource groups where the AI Hub and its components will be deployed.
- For APIM fetching secrets from Key Vault, the APIM's Managed Identity (if System Assigned) or User Assigned Identity will need "Key Vault Secrets User" or "Get" permissions on the Key Vault.

4. OpenAI Model Integration:

- The azure_rm_api_management_api for OpenAI is a placeholder. You'll need to update service_url to the actual endpoint of your Azure OpenAI Service instance (e.g., <https://<your-openai-resource-name>.openai.azure.com/>).
- **API Key Management:** The API key for Azure OpenAI should be stored securely, ideally in Azure Key Vault. The APIM module has a placeholder for azure_rm_api_management_named_value and azure_rm_user_assigned_identity to demonstrate how APIM can fetch this key from Key Vault. **Never hardcode API keys in Terraform.**

5. Custom Policies:

- The xml_content in azure_rm_api_management_api and azure_rm_api_management_product is where you define your custom policies (e.g., rate limiting, JWT validation, IP filtering, request/response transformations).
- For complex policies, consider using separate .xml files and reading them into Terraform using file() function, or dynamic generation as shown in some samples.
- **Named Values:** Use azure_rm_api_management_named_value for policy expressions that need external values (like API keys, tenant IDs, etc.).

6. Self-Service Pipeline for Application Teams:

- This is distinct from the infrastructure deployment.
- **Process:**
 1. Application team initiates a request (e.g., via a custom web portal, Azure DevOps pipeline, or service catalog).
 2. The request triggers an Azure Function or Logic App.
 3. This automation interacts with:
 - **Azure API Management:**
 - Creates a new azure_rm_api_management_subscription for the application team.
 - Assigns the subscription to the relevant

azurerm_api_management_product (e.g., "OpenAI Access Product").

- Optionally, creates a new azurerm_api_management_user for the application owner.
- Generates and provides subscription keys to the application team.

- **Azure AI Hub (if direct access is needed, unlikely):** Grants necessary permissions (e.g., Reader, or specific AI service roles) to the application's Managed Identity or Service Principal.

4. **Approvals:** Implement approval steps in Azure DevOps Environments or Power Automate flows before provisioning API subscriptions.

- **Terraform for App Onboarding (if part of self-service):** While the infrastructure is Terraform, the self-service could be a lightweight pipeline that *also* uses Terraform to manage APIM subscriptions and users, if you want full IaC for onboarding. This would be a separate, smaller Terraform project.

7. **Cost Management & Monitoring Details:**

- **Cost Management:** The provided budget resource is a basic example. You can add more granular budgets (per resource group), cost anomaly alerts, and export cost data to storage accounts for Power BI reporting.
- **Monitoring:** The Event Hub, Function App, Log Analytics, and Application Insights provide a strong foundation. The Function App would contain the logic to process Event Hub data and push it to Power BI or other destinations.
- **Power BI:** Terraform doesn't directly manage Power BI datasets or dashboards. You'd use the Power BI API or Power BI Desktop/Service for that. The function app is where you'd implement the data ingestion into Power BI.

8. **VNet & Subnets:**

- The current setup assumes existing VNet and subnets. Ensure these subnets are properly sized and have Network Security Groups (NSGs) configured to allow necessary traffic for APIM and Private Endpoints.
- APIM VNet integration has specific subnet requirements (e.g., delegation). Review Azure documentation.

9. **Naming Conventions:** Implement consistent naming conventions for your resources. The provided code uses basic conventions; customize them to your organization's standards.

This comprehensive guide and Terraform code will provide a solid foundation for deploying your Azure AI Hub Landing Zone Accelerator. Remember to test thoroughly in development environments before moving to production.