NYCPS TMS - Expanded Terraform Module Examples

This document provides more detailed, representative Terraform code examples for key AWS GovCloud services identified in the architecture. Place these within appropriately named subdirectories inside your main `modules/` directory.

> **CRITICAL:** This is example code. Customize names, CIDRs, ARNs, IAM policies, security groups, instance types, KMS keys, tags, and all other configuration parameters according to NYCPS requirements, security policies, and best practices before deploying to any environment. Always run `terraform plan` and review carefully.

Module: Secure S3 Bucket (`modules/s3_bucket/`)

Creates a private S3 bucket with versioning, encryption, access logging, and public access block.

`variables.tf`

```hcl
variable "bucket_name_prefix" {
  description = "Prefix for the S3 bucket name (will have env/random suffi
  type        = string
}

variable "environment_name" {
  description = "Environment name (e.g., dev, prod)"
  type        = string
}

variable "kms_key_arn" {
  description = "Optional: KMS Key ARN for SSE-KMS encryption"
  type        = string
  default     = null
}

variable "access_log_bucket_name" {
  description = "Name of the S3 bucket where access logs should be deliver
  type        = string
  default     = null # Set to enable logging
}

variable "lifecycle_rules" {
  description = "List of lifecycle rule objects (e.g., for transitions to
  type        = any # Complex type, define structure or use specific objec
  default     = []
}

variable "tags" {
  description = "Map of tags to assign to the bucket"
  type        = map(string)
  default     = {}
}
```

`main.tf`

```
resource "random_id" "suffix" {
  byte_length = 4
}

resource "aws_s3_bucket" "this" {
  bucket = "${var.bucket_name_prefix}-${var.environment_name}-${random_id.

  tags = var.tags
}

resource "aws_s3_bucket_versioning" "this" {
  bucket = aws_s3_bucket.this.id
  versioning_configuration {
    status = "Enabled"
  }
}

resource "aws_s3_bucket_server_side_encryption_configuration" "this" {
  bucket = aws_s3_bucket.this.id

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm     = var.kms_key_arn == null ? "AES256" : "aws:kms"
      kms_master_key_id = var.kms_key_arn
    }
  }
}

resource "aws_s3_bucket_public_access_block" "this" {
  bucket                  = aws_s3_bucket.this.id
  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
```

```
}

resource "aws_s3_bucket_logging" "this" {
  count = var.access_log_bucket_name == null ? 0 : 1

  bucket = aws_s3_bucket.this.id

  target_bucket = var.access_log_bucket_name
  target_prefix = "log/${aws_s3_bucket.this.bucket}/"
}

resource "aws_s3_bucket_lifecycle_configuration" "this" {
  count = length(var.lifecycle_rules) > 0 ? 1 : 0

  bucket = aws_s3_bucket.this.id

  dynamic "rule" {
    for_each = var.lifecycle_rules
    content {
      id     = rule.value.id
      status = lookup(rule.value, "status", "Enabled")

      dynamic "filter" { # Optional filter block
        for_each = lookup(rule.value, "filter", null) != null ? [1] : []
        content {
          prefix = lookup(rule.value.filter, "prefix", null)
          # Add tags filter if needed
        }
      }

      dynamic "transition" {
        for_each = lookup(rule.value, "transition", [])
        content {
          days          = lookup(transition.value, "days", null)
          storage_class = transition.value.storage_class
        }
      }
```

```
      dynamic "expiration" {
         for_each = lookup(rule.value, "expiration", null) != null ? [1] :
         content {
            days = lookup(rule.value.expiration, "days", null)
            # expired_object_delete_marker = true # for versioning
         }
      }
       # Add noncurrent_version_transition/expiration if needed
    }
  }
}
```

`outputs.tf`

```
output "bucket_id" {
  description = "The name of the bucket"
  value       = aws_s3_bucket.this.id
}
output "bucket_arn" {
  description = "The ARN of the bucket"
  value       = aws_s3_bucket.this.arn
}
```

Module: IAM Role (`modules/iam_role/`)

Creates a generic IAM role for an AWS service (e.g., Lambda, EC2, ECS Task) with specified policies.

`variables.tf`

```
variable "role_name" { type = string }
variable "description" { type = string; default = "IAM role" }
variable "assume_role_policy_json" {
  description = "JSON policy document allowing services (e.g., lambda.amaz
  type        = string
}
variable "managed_policy_arns" {
  description = "List of ARNs for AWS Managed Policies to attach"
  type        = list(string)
  default     = []
}
variable "inline_policy_json" {
  description = "Optional: JSON policy document for an inline policy"
  type        = string
  default     = null
}
variable "inline_policy_name" {
  description = "Name for the inline policy, required if inline_policy_jso
  type        = string
  default     = "inline-policy"
}
variable "tags" { type = map(string); default = {} }
```

`main.tf`

```
resource "aws_iam_role" "this" {
  name               = var.role_name
  description        = var.description
  assume_role_policy = var.assume_role_policy_json
  tags               = var.tags
```

```
}

# Attach Managed Policies
resource "aws_iam_role_policy_attachment" "managed" {
  count      = length(var.managed_policy_arns)
  role       = aws_iam_role.this.name
  policy_arn = var.managed_policy_arns[count.index]
}

# Attach Inline Policy (if provided)
resource "aws_iam_role_policy" "inline" {
  count  = var.inline_policy_json != null ? 1 : 0
  name   = var.inline_policy_name
  role   = aws_iam_role.this.id
  policy = var.inline_policy_json
}
```

`outputs.tf`

```
output "role_arn" {
  description = "ARN of the created IAM role"
  value       = aws_iam_role.this.arn
}
output "role_name" {
  description = "Name of the created IAM role"
  value       = aws_iam_role.this.name
}
```

> **IAM Security:** Craft `assume_role_policy_json` and specific policy documents (passed via `inline_policy_json` or created as separate `aws_iam_policy` resources and passed via `managed_policy_arns`) with extreme care, adhering strictly to least privilege. Avoid wildcard (`*`) permissions wherever possible.

Module: Lambda Function (`modules/lambda_function/`)

Creates an AWS Lambda function from a deployment package in S3.

`variables.tf`

```
variable "function_name" { type = string }
variable "description" { type = string; default = "Lambda function" }
variable "handler" { type = string } # e.g., "index.handler" for Node.js
variable "runtime" { type = string } # e.g., "nodejs18.x", "python3.11"
variable "iam_role_arn" { type = string } # ARN from the IAM role module
variable "s3_bucket" { type = string } # Bucket where deployment package s
variable "s3_key" { type = string } # Key (path) to the .zip file in S3
variable "s3_object_version" { type = string; default = null } # Optional
variable "memory_size" { type = number; default = 128 } # MB
variable "timeout" { type = number; default = 30 } # Seconds
variable "environment_variables" { type = map(string); default = {} } # No
variable "vpc_subnet_ids" { type = list(string); default = null } # Requir
variable "vpc_security_group_ids" { type = list(string); default = null }
variable "kms_key_arn_environment" { type = string; default = null } # For
variable "tags" { type = map(string); default = {} }
```

`main.tf`

```
resource "aws_lambda_function" "this" {
  function_name    = var.function_name
  description      = var.description
```

```
  role             = var.iam_role_arn
  handler          = var.handler
  runtime          = var.runtime
  memory_size      = var.memory_size
  timeout          = var.timeout

  s3_bucket        = var.s3_bucket
  s3_key           = var.s3_key
  s3_object_version= var.s3_object_version
  # source_code_hash = filebase64sha256("path/to/your/deployment.zip") # U

  dynamic "environment" {
    for_each = length(keys(var.environment_variables)) > 0 ? [1] : []
    content {
      variables = var.environment_variables
    }
  }

  dynamic "vpc_config" {
    for_each = var.vpc_subnet_ids != null ? [1] : []
    content {
      subnet_ids         = var.vpc_subnet_ids
      security_group_ids = var.vpc_security_group_ids
    }
  }

  kms_key_arn = var.kms_key_arn_environment

  tags = var.tags

  # Ensure the IAM role exists before creating the function
  depends_on = [aws_iam_role.this] # Assuming role is defined in the same
}

# Optional: CloudWatch Log Group for the Lambda function
resource "aws_cloudwatch_log_group" "this" {
  name              = "/aws/lambda/${var.function_name}"
```

```
  retention_in_days = 14 # TODO: Adjust retention
  # kms_key_id = var.kms_key_arn # Optional: Encrypt log group
}
```

`outputs.tf`

```
output "function_arn" {
  description = "ARN of the Lambda function"
  value       = aws_lambda_function.this.arn
}
output "function_name" {
  description = "Name of the Lambda function"
  value       = aws_lambda_function.this.function_name
}
```

> **Deployment Packages:** The Lambda function code needs to be packaged (e.g., as a .zip file) and uploaded to the specified S3 bucket, typically as part of your CI/CD pipeline before Terraform runs `apply`. Terraform references the S3 object.

Module: ECS Fargate Service (`modules/ecs_fargate_service/`)

Creates an ECS Cluster (optional, can be shared), Task Definition, and Fargate Service with basic networking and load balancing.

`variables.tf`

```
variable "service_name" { type = string }
variable "environment_name" { type = string }
variable "ecs_cluster_arn" { type = string } # ARN of an existing shared c
variable "task_cpu" { type = number; default = 256 } # CPU units (1024 = 1
variable "task_memory" { type = number; default = 512 } # Memory in MiB
variable "container_image" { type = string } # ECR image URI (e.g., ACCOUN
variable "container_port" { type = number } # Port the container listens c
variable "desired_count" { type = number; default = 2 } # Minimum number c
variable "task_execution_role_arn" { type = string } # Role for ECS agent
variable "task_role_arn" { type = string; default = null } # Optional role
variable "vpc_id" { type = string }
variable "private_subnet_ids" { type = list(string) }
variable "security_group_ids" { type = list(string) } # SG allowing traffi
variable "alb_target_group_arn" { type = string; default = null } # Optior
variable "container_environment_variables" { type = list(object({ name = s
variable "container_secrets" { type = list(object({ name = string, valueFr
variable "tags" { type = map(string); default = {} }
```

`main.tf`

```
# Optional: Create ECS Cluster if not using a shared one
# resource "aws_ecs_cluster" "this" { ... }

# CloudWatch Log Group for the container
resource "aws_cloudwatch_log_group" "this" {
  name              = "/ecs/${var.service_name}-${var.environment_name}"
  retention_in_days = 14 # TODO: Adjust
}

resource "aws_ecs_task_definition" "this" {
```

```
  family                   = "${var.service_name}-${var.environment_name}"
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = var.task_cpu
  memory                   = var.task_memory
  execution_role_arn       = var.task_execution_role_arn
  task_role_arn            = var.task_role_arn

  container_definitions = jsonencode([
    {
      name      = var.service_name
      image     = var.container_image
      cpu       = var.task_cpu # Often same as task CPU for single contair
      memory    = var.task_memory # Often same as task Memory
      essential = true
      portMappings = [
        {
          containerPort = var.container_port
          hostPort      = var.container_port # HostPort must match contair
          protocol      = "tcp"
        }
      ]
      environment = var.container_environment_variables
      secrets     = var.container_secrets
      logConfiguration = {
        logDriver = "awslogs"
        options = {
          "awslogs-group"         = aws_cloudwatch_log_group.this.name
          "awslogs-region"        = data.aws_region.current.name
          "awslogs-stream-prefix" = var.service_name
        }
      }
    }
    # Add more containers if needed (e.g., sidecars)
  ])

  tags = var.tags
```

```
}

resource "aws_ecs_service" "this" {
  name            = "${var.service_name}-${var.environment_name}"
  cluster         = var.ecs_cluster_arn
  task_definition = aws_ecs_task_definition.this.arn
  desired_count   = var.desired_count
  launch_type     = "FARGATE"

  network_configuration {
    subnets         = var.private_subnet_ids
    security_groups = var.security_group_ids
    assign_public_ip = false # Typically false for private subnets
  }

  dynamic "load_balancer" {
    for_each = var.alb_target_group_arn != null ? [1] : []
    content {
      target_group_arn = var.alb_target_group_arn
      container_name   = var.service_name
      container_port   = var.container_port
    }
  }

  # Optional: Deployment configuration (rolling update, blue/green)
  # Optional: Service discovery configuration (Cloud Map)
  # Optional: Auto Scaling configuration (aws_appautoscaling_target, aws_a

  tags = var.tags
}

data "aws_region" "current" {}
```

`outputs.tf`

```
output "service_name" {
  description = "Name of the ECS service"
  value       = aws_ecs_service.this.name
}
# ... other outputs
```

Module: DynamoDB Table (`modules/dynamodb_table/`)

Creates a DynamoDB table with basic configuration.

`variables.tf`

```
variable "table_name" { type = string }
variable "billing_mode" { type = string; default = "PAY_PER_REQUEST" } # C
variable "hash_key" { type = string }
variable "range_key" { type = string; default = null }
variable "attributes" { # Define attributes used in keys/indexes
  type = list(object({
    name = string
    type = string # S=String, N=Number, B=Binary
  }))
}
variable "enable_pitr" { type = bool; default = true } # Point-in-time rec
variable "kms_key_arn" { type = string; default = null } # For CMK encrypt
```

```
variable "tags" { type = map(string); default = {} }
# TODO: Add variables for GSIs, LSIs, Stream Specification if needed
```

`main.tf`

```
resource "aws_dynamodb_table" "this" {
  name         = var.table_name
  billing_mode = var.billing_mode
  hash_key     = var.hash_key
  range_key    = var.range_key

  dynamic "attribute" {
    for_each = var.attributes
    content {
      name = attribute.value.name
      type = attribute.value.type
    }
  }

  point_in_time_recovery {
    enabled = var.enable_pitr
  }

  server_side_encryption {
    enabled     = true
    kms_key_arn = var.kms_key_arn # If null, uses AWS owned key
  }

  # TODO: Add blocks for global_secondary_index, local_secondary_index, st
  # ttl { enabled = false } # Optional TTL configuration
```

```
  tags = var.tags
}
```

`outputs.tf`

```
output "table_name" { value = aws_dynamodb_table.this.name }
output "table_arn" { value = aws_dynamodb_table.this.arn }
```

Module: Kinesis Data Stream (`modules/kinesis_stream/`)

Creates a Kinesis Data Stream.

`variables.tf`

```
variable "stream_name" { type = string }
variable "shard_count" { type = number; default = 1 }
variable "retention_period" { type = number; default = 24 } # Hours
variable "kms_key_id" { type = string; default = null } # ARN for SSE-KMS
variable "tags" { type = map(string); default = {} }
```

`main.tf`

```
resource "aws_kinesis_stream" "this" {
  name             = var.stream_name
  shard_count      = var.shard_count
  retention_period = var.retention_period

  # Use "PROVISIONED" (default) or "ON_DEMAND"
  # shard_level_metrics = ["IncomingBytes", "OutgoingBytes"]

  stream_mode_details {
     stream_mode = "PROVISIONED" # Or "ON_DEMAND"
  }

  encryption_type = var.kms_key_id == null ? "NONE" : "KMS"
  kms_key_id      = var.kms_key_id

  tags = var.tags
}
```

`outputs.tf`

```
output "stream_name" { value = aws_kinesis_stream.this.name }
output "stream_arn" { value = aws_kinesis_stream.this.arn }
```

Module: SNS Topic (`modules/sns_topic/`)

Creates an SNS topic.

`variables.tf`

```
variable "topic_name" { type = string }
variable "kms_master_key_id" { type = string; default = null } # Optional
variable "tags" { type = map(string); default = {} }
```

`main.tf`

```
resource "aws_sns_topic" "this" {
  name              = var.topic_name
  kms_master_key_id = var.kms_master_key_id # Use "alias/aws/sns" for AWS-
  tags              = var.tags
  # TODO: Add aws_sns_topic_policy if specific cross-account or service ac
}
```

`outputs.tf`

```
output "topic_arn" { value = aws_sns_topic.this.arn }
```

Module: SQS Queue (`modules/sqs_queue/`)

Creates a standard SQS queue.

`variables.tf`

```
variable "queue_name" { type = string }
variable "is_fifo_queue" { type = bool; default = false }
variable "visibility_timeout_seconds" { type = number; default = 30 }
variable "message_retention_seconds" { type = number; default = 345600 } #
variable "kms_master_key_id" { type = string; default = null } # Use "alia
variable "kms_data_key_reuse_period_seconds" { type = number; default = 30
variable "dead_letter_queue_arn" { type = string; default = null } # ARN o
variable "max_receive_count" { type = number; default = 5 } # Used with DL
variable "tags" { type = map(string); default = {} }
```

`main.tf`

```
resource "aws_sqs_queue" "this" {
  name                              = var.is_fifo_queue ? "${var.queue_nam
  fifo_queue                        = var.is_fifo_queue
  content_based_deduplication       = var.is_fifo_queue ? true : false # C
  visibility_timeout_seconds        = var.visibility_timeout_seconds
  message_retention_seconds         = var.message_retention_seconds
  kms_master_key_id                 = var.kms_master_key_id
  kms_data_key_reuse_period_seconds = var.kms_data_key_reuse_period_second

  redrive_policy = var.dead_letter_queue_arn == null ? null : jsonencode({
    deadLetterTargetArn = var.dead_letter_queue_arn
    maxReceiveCount     = var.max_receive_count
  })

  tags = var.tags
```

```
  # TODO: Add aws_sqs_queue_policy if needed
}
```

`outputs.tf`

```
output "queue_url" { value = aws_sqs_queue.this.id } # Note: id attribute
output "queue_arn" { value = aws_sqs_queue.this.arn }
```

Module: API Gateway (HTTP API Example) (`modules/api_gateway/`)

Creates a basic HTTP API Gateway with Lambda integration.

`variables.tf`

```
variable "api_name" { type = string }
variable "lambda_integration_uri" { type = string } # Invoke ARN of the ta
variable "route_key" { type = string; default = "ANY /{proxy+}" } # Defaul
variable "tags" { type = map(string); default = {} }
# TODO: Add variables for custom domain, authorizers, CORS, logging, etc.
```

`main.tf`

```
resource "aws_apigatewayv2_api" "this" {
  name          = var.api_name
  protocol_type = "HTTP"
  description   = "HTTP API for ${var.api_name}"
  # cors_configuration { ... } # TODO: Configure CORS if needed
  tags          = var.tags
}

resource "aws_apigatewayv2_integration" "lambda" {
  api_id             = aws_apigatewayv2_api.this.id
  integration_type   = "AWS_PROXY"
  integration_method = "POST" # Always POST for Lambda proxy
  integration_uri    = var.lambda_integration_uri
  payload_format_version = "2.0" # Use payload format 2.0 for HTTP APIs
}

resource "aws_apigatewayv2_route" "proxy" {
  api_id    = aws_apigatewayv2_api.this.id
  route_key = var.route_key
  target    = "integrations/${aws_apigatewayv2_integration.lambda.id}"
  # TODO: Add authorizer configuration if needed
  # authorization_type = "JWT" / "AWS_IAM" / "CUSTOM"
  # authorizer_id = aws_apigatewayv2_authorizer.this.id
}

resource "aws_apigatewayv2_stage" "default" {
  api_id = aws_apigatewayv2_api.this.id
  name   = "$default" # Creates a default stage accessible at the base inv
  auto_deploy = true

  # TODO: Configure access logging, throttling, custom domain mapping
  # access_log_settings { ... }
  # default_route_settings { ... }
  # domain_name { ... }
```

```
}

# Permissions for API Gateway to invoke Lambda
resource "aws_lambda_permission" "api_gw" {
  statement_id  = "AllowAPIGatewayInvoke"
  action        = "lambda:InvokeFunction"
  function_name = var.lambda_integration_uri # Use the function name part
  principal     = "apigateway.amazonaws.com"

  # Restrict to the specific API Gateway API
  source_arn = "${aws_apigatewayv2_api.this.execution_arn}/*/*"
}
```

`outputs.tf`

```
output "api_endpoint" {
  description = "The invoke URL for the API Gateway stage"
  value       = aws_apigatewayv2_stage.default.invoke_url
}
output "api_id" {
  description = "The ID of the API Gateway"
  value       = aws_apigatewayv2_api.this.id
}
```

Module: CloudWatch Alarm (`modules/cloudwatch_alarm/`)

Creates a basic CloudWatch alarm based on a metric.

`variables.tf`

```
variable "alarm_name" { type = string }
variable "comparison_operator" { type = string } # e.g., "GreaterThanOrEqu
variable "evaluation_periods" { type = number } # e.g., 1
variable "metric_name" { type = string } # e.g., "Errors"
variable "namespace" { type = string } # e.g., "AWS/Lambda"
variable "period" { type = number } # Seconds, e.g., 300 (5 minutes)
variable "statistic" { type = string } # e.g., "Sum", "Average"
variable "threshold" { type = number }
variable "alarm_description" { type = string; default = null }
variable "dimensions" { type = map(string); default = {} } # e.g., { Funct
variable "alarm_actions" { type = list(string); default = [] } # List of S
variable "ok_actions" { type = list(string); default = [] } # Optional: ac
```

`main.tf`

```
resource "aws_cloudwatch_metric_alarm" "this" {
  alarm_name          = var.alarm_name
  comparison_operator = var.comparison_operator
  evaluation_periods  = var.evaluation_periods
  metric_name         = var.metric_name
  namespace           = var.namespace
  period              = var.period
  statistic           = var.statistic
  threshold           = var.threshold
  alarm_description   = var.alarm_description
  dimensions          = length(keys(var.dimensions)) > 0 ? var.dimensions

  alarm_actions = var.alarm_actions
  ok_actions    = var.ok_actions
```

```
  # treat_missing_data = "missing" # Other options: "ignore", "breaching",
}
```

`outputs.tf`

```
output "alarm_arn" { value = aws_cloudwatch_metric_alarm.this.arn }
output "alarm_name" { value = aws_cloudwatch_metric_alarm.this.alarm_name
```

Module: VPC Interface Endpoint (`modules/vpc_endpoint/`)

Creates a VPC Interface Endpoint for accessing AWS services privately.

`variables.tf`

```
variable "vpc_id" { type = string }
variable "service_name" { type = string } # e.g., "com.amazonaws.us-gov-we
variable "subnet_ids" { type = list(string) } # Subnets where the ENI will
variable "security_group_ids" { type = list(string) } # SG allowing HTTPS
variable "private_dns_enabled" { type = bool; default = true }
variable "tags" { type = map(string); default = {} }
```

`main.tf`

```
resource "aws_vpc_endpoint" "this" {
  vpc_id              = var.vpc_id
  service_name        = var.service_name
  vpc_endpoint_type   = "Interface"
  subnet_ids          = var.subnet_ids
  security_group_ids  = var.security_group_ids
  private_dns_enabled = var.private_dns_enabled
  tags                = var.tags
}
```

`outputs.tf`

```
output "endpoint_id" { value = aws_vpc_endpoint.this.id }
output "dns_entries" { value = aws_vpc_endpoint.this.dns_entry }
output "network_interface_ids" { value = aws_vpc_endpoint.this.network_int
```

Next Steps & Implementation

1. **Create Directories:** Set up the `modules/` and `environments/` directory structure shown in Step 1.

2. **Populate Modules:** Copy the example code above into the corresponding module directories (e.g., `modules/s3_bucket/`, `modules/iam_role/`).

3. **Develop Remaining Modules:** Using the provided examples as a template, develop the Terraform code (`variables.tf`, `main.tf`, `outputs.tf`) for all other required components identified in the architecture (e.g., ElastiCache, Redshift, MSK, Glue, additional IAM policies, specific Security Groups per tier, Load Balancers, CloudFront, etc.). Remember to focus on modularity and parameterization.

4. **Create Environment Files:** Create the `main.tf`, `variables.tf`, `outputs.tf`, and `.tfvars` files within each directory under `environments/` (dev, qa, prod, etc.).

5. **Instantiate Modules:** In each `environments/*/main.tf`, add `module` blocks calling the necessary modules from your `modules/` directory, passing outputs from prerequisite modules (like networking) as inputs to dependent modules (like RDS or ECS).

6. **Define Variables:** Fill in the `.tfvars` files for each environment with the appropriate configuration values, ensuring sensitive data is handled securely (e.g., referencing Secrets Manager ARNs).

7. **Initialize, Plan, Apply (Iteratively):** Follow the workflow in Step 6, starting with the `dev` environment. Initialize, plan carefully, review the plan, and apply. Address any errors. Test the provisioned resources. Repeat iteratively as you build out more components and refine configurations.

8. **Version Control:** Commit all code frequently to your Git repository.

9. **Integrate with CI/CD:** Once the manual workflow is stable, integrate the `plan` and `apply` steps into your CI/CD pipeline for automated environment management, including appropriate approval steps for Staging and Production environments.