포팅메뉴얼

1. AWS CLI 설치

- https://docs.aws.amazon.com/ko_kr/cli/latest/userguide/getting-started-install.html
- 윈도우 다운로드
- aws configure 로 계정 로그인

2. Terraform 설치

- https://developer.hashicorp.com/terraform/install
- 윈도우 다운로드

3. Node 설치

- https://nodejs.org/en/download/package-manager
- npm init

4. 테라폼 작성

- chat main.tf

```
resource "aws_sqs_queue" "chat_quotation_calculation_trigger"
  name = "chat-quotation-calculation-trigger"
}
```

```
resource "aws_dynamodb_table" "chat_sessions" {
  name         = "chat-app-CustomerChatSessions"
  billing_mode = "PAY_PER_REQUEST"

  # 테이블 키와 속성을 정확히 입력해야 합니다.
  hash_key     = "orderId"  # 실제 테이블의 파티션 키 이름으로 변경

  attribute {
    name = "orderId"        # 파티션 키 이름
    type = "S"                          # 파티션 키 유형 (예: S
  }
```

- classification

```
#sqs
resource "aws_sqs_queue" "information_intergrity_verification
  name = "information-integrity-verification-trigger"
}

resource "aws_sqs_queue" "mail_extraction_trigger" {
  name = "mail-extraction-trigger"
}

resource "aws_sqs_queue" "mail_save_trigger" {
  name = "mail-save-trigger"
}

#ecr
resource "aws_ecr_repository" "mail_extraction" {
  name = "mail-extraction"
  image_tag_mutability = "MUTABLE"
}

## dynamodb
resource "aws_dynamodb_table" "mail_db" {
  name           = "mail-db"
```

```
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "receiver"            # 파티션 키 설정
  range_key      = "received_date"       # 정렬 키 설정

  attribute {
    name = "receiver"
    type = "S"
  }

  attribute {
    name = "received_date"
    type = "S"
  }

  stream_enabled    = true
  stream_view_type  = "NEW_AND_OLD_IMAGES"

  tags = {
    Environment = "dev"
  }
}
```

- cogonito main.tf

```
# resource "aws_cognito_user_pool_domain" "main" {
#   domain       = "busybeemail"  # "https://"를 제외하고, 서브도
#   user_pool_id = aws_cognito_user_pool.cognito_pool.id
# }

# resource "aws_cognito_user_pool" "cognito_pool" {
#   name = "cognito_pool"

#   # 비밀번호 정책 설정
#   password_policy {
#     minimum_length    = 8
#     require_lowercase = true
#     require_uppercase = true
```

```
#     require_numbers   = true
#     require_symbols   = true
#   }

#   # MFA 설정 (선택 사항)
#   mfa_configuration = "OPTIONAL"
#   software_token_mfa_configuration {
#     enabled = false
#   }

#   # 이메일 인증 옵션
#   auto_verified_attributes = ["email"]
# }

# resource "aws_cognito_user_pool_client" "app_client" {
#   name            = "app_client"
#   user_pool_id    = aws_cognito_user_pool.cognito_pool.id
#   generate_secret = false

#   # 허용된 콜백 URL 설정
#   callback_urls = [
#     "https://busybeemail.net",
#   ]
# }


resource "aws_cognito_user_pool" "cognito_pool" {
  name = "cognito-pool"

  # 비밀번호 정책 설정
  password_policy {
    minimum_length    = 8
    require_lowercase = true
    require_uppercase = true
    require_numbers   = true
    require_symbols   = true
  }
```

```
  # MFA 설정 (선택 사항)
  mfa_configuration = "OPTIONAL"
  software_token_mfa_configuration {
    enabled = false
  }


  # 이메일 인증 옵션
  auto_verified_attributes = ["email"]

  schema {
    attribute_data_type = "String"
    name                = "email"
    required            = true
    mutable             = false
  }

  verification_message_template {
    default_email_option = "CONFIRM_WITH_LINK"
  }
}

resource "aws_cognito_user_pool_domain" "main" {
  domain       = "busybeemail-unique"  # 고유한 도메인 이름
  user_pool_id = aws_cognito_user_pool.cognito_pool.id
}

resource "aws_cognito_user_pool_client" "app_client" {
  name                         = "app_client"
  user_pool_id                 = aws_cognito_user_pool.cognito_po
  generate_secret              = false

  # 허용된 콜백 URL 및 로그아웃 URL 설정
  callback_urls = [
    "https://busybeemail.net/",  # 콜백 URL
  ]
  logout_urls = [
    "https://busybeemail.net/logout",  # 로그아웃 URL
```

```
  ]

  # OAuth 설정
  allowed_oauth_flows       = ["implicit"]  # 암시적 권한 부여 추
  allowed_oauth_scopes      = ["email", "openid", "profile"]
  allowed_oauth_flows_user_pool_client = true
  supported_identity_providers = ["COGNITO"]
}



// mqtt
// mqtt

# IoT Policy 생성 (이전 aws_iot_policy 삭제)
data "aws_iam_policy_document" "iot_policy_doc" {
  statement {
    actions   = ["iot:Connect", "iot:Publish", "iot:Subscribe
    resources = ["*"]
  }
}

# IAM 인라인 정책을 통해 IoT 권한을 Cognito 역할에 연결
resource "aws_iam_role_policy" "iot_policy" {
  name   = "Cognito_IotPolicy"
  role   = aws_iam_role.unauthenticated_role.id
  policy = data.aws_iam_policy_document.iot_policy_doc.json
}

# Cognito Identity Pool 생성
resource "aws_cognito_identity_pool" "identity_pool" {
  identity_pool_name               = "MyIotIdentityPool"
  allow_unauthenticated_identities = true
}

# Cognito Role 생성 및 설정
resource "aws_iam_role" "unauthenticated_role" {
  name = "Cognito_UnAuth_Role"
```

```
  assume_role_policy = jsonencode({
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "Federated": "cognito-identity.amazonaws.com"
        },
        "Action": "sts:AssumeRoleWithWebIdentity",
        "Condition": {
          "StringEquals": {
            "cognito-identity.amazonaws.com:aud": aws_cognito
          },
          "ForAnyValue:StringLike": {
            "cognito-identity.amazonaws.com:amr": "unauthenti
          }
        }
      }
    ]
  })
}

# Identity Pool에 역할 연결
resource "aws_cognito_identity_pool_roles_attachment" "identi
  identity_pool_id = aws_cognito_identity_pool.identity_pool.
  roles = {
    "unauthenticated" = aws_iam_role.unauthenticated_role.arn
  }
}

output "identity_pool_id" {
  value = aws_cognito_identity_pool.identity_pool.id
}
```

- information_verification main.tf

```
#sqs
resource "aws_sqs_queue" "quotation_calculation_trigger" {
    name = "quotation-calculation-trigger"
}

## dynamodb
resource "aws_dynamodb_table" "example_table" {
  name           = "estimate"
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "Id"

  attribute {
    name = "Id"
    type = "S"
  }

  stream_enabled = true
  stream_view_type = "NEW_AND_OLD_IMAGES"

  tags = {
    Environment = "dev"
  }
}


## ecr
resource "aws_ecr_repository" "information_intergrity_verific
    name = "information-intergrity-verification"
    image_tag_mutability = "MUTABLE"
}

resource "aws_ecr_repository" "quotation_calculation" {
    name = "quotation-calculation"
    image_tag_mutability = "MUTABLE"
}

resource "aws_ecr_repository" "save_data" {
  name = "save-data"
```

```
  image_tag_mutability = "MUTABLE"
}

resource "aws_ecr_repository" "send_quote_email" {
    name = "send-quote-mail"
    image_tag_mutability = "MUTABLE"
}

resource "aws_ecr_repository" "information_mail_request" {
    name = "information-mail-request"
    image_tag_mutability = "MUTABLE"
}


## sns sqs 연결 모듈
## 정보 검증 성공
module "save_data" {
  source  = "terraform-aws-modules/sns/aws"
  version = ">= 5.0"

  name = "save-data"

  topic_policy_statements = {
    sqs = {
      sid       = "SQSSubscribe"
      actions   = ["sns:Subscribe", "sns:Receive"]
      principals = [{
        type        = "AWS"
        identifiers = ["*"]
      }]
      conditions = [{
        test     = "StringLike"
        variable = "sns:Endpoint"
        values   = [module.save_data_sqs.queue_arn, module.se
      }]
    }
  }
```

```
  subscriptions = {
    save_data_sqs = {
      protocol = "sqs"
      endpoint = module.save_data_sqs.queue_arn
    }
    send_quote_mail_sqs = {
      protocol = "sqs"
      endpoint = module.send_quote_mail_sqs.queue_arn
    }
  }

  tags = {
    Environment = "dev"
  }
}

module "save_data_sqs" {
  source = "terraform-aws-modules/sqs/aws"

  name = "save-data-trigger"

  create_queue_policy = true
  queue_policy_statements = {
    sns = {
      sid     = "SNSPublish"
      actions = ["sqs:SendMessage"]

      principals = [
        {
          type        = "Service"
          identifiers = ["sns.amazonaws.com"]
        }
      ]

      conditions = [{
        test     = "ArnEquals"
        variable = "aws:SourceArn"
        values   = [module.save_data.topic_arn]
```

```
      }]
    }
  }

  tags = {
    Environment = "dev"
  }
}

module "send_quote_mail_sqs" {
  source = "terraform-aws-modules/sqs/aws"

  name = "send-quote-mail-trigger"

  create_queue_policy = true
  queue_policy_statements = {
    sns = {
      sid     = "SNSPublish"
      actions = ["sqs:SendMessage"]

      principals = [
        {
          type        = "Service"
          identifiers = ["sns.amazonaws.com"]
        }
      ]

      conditions = [{
        test     = "ArnEquals"
        variable = "aws:SourceArn"
        values   = [module.save_data.topic_arn]
      }]
    }
  }

  tags = {
    Environment = "dev"
  }
```

```
}

## 정보 검증 실패
module "incorrect_information" {
  source  = "terraform-aws-modules/sns/aws"
  version = ">= 5.0"

  name = "incorrect-information"

  topic_policy_statements = {
    sqs = {
      sid       = "SQSSubscribe"
      actions   = ["sns:Subscribe", "sns:Receive"]
      principals = [{
        type        = "AWS"
        identifiers = ["*"]
      }]
      conditions = [{
        test     = "StringLike"
        variable = "sns:Endpoint"
        values   = [module.save_incorrect_information_sqs.que
      }]
    }
  }

  subscriptions = {
    save_incorrect_information_sqs = {
      protocol = "sqs"
      endpoint = module.save_incorrect_information_sqs.queue_
    }
    information_mail_request_sqs = {
      protocol = "sqs"
      endpoint = module.information_mail_request_sqs.queue_ar
    }
  }

  tags = {
    Environment = "dev"
```

```
  }
}

module "save_incorrect_information_sqs" {
  source = "terraform-aws-modules/sqs/aws"

  name = "save-incorrect-information-trigger"

  create_queue_policy = true
  queue_policy_statements = {
    sns = {
      sid     = "SNSPublish"
      actions = ["sqs:SendMessage"]

      principals = [
        {
          type        = "Service"
          identifiers = ["sns.amazonaws.com"]
        }
      ]

      conditions = [{
        test     = "ArnEquals"
        variable = "aws:SourceArn"
        values   = [module.incorrect_information.topic_arn]
      }]
    }
  }

  tags = {
    Environment = "dev"
  }
}

module "information_mail_request_sqs" {
  source = "terraform-aws-modules/sqs/aws"

  name = "information-mail-request-trigger"
```

```
  create_queue_policy = true
  queue_policy_statements = {
    sns = {
      sid     = "SNSPublish"
      actions = ["sqs:SendMessage"]

      principals = [
        {
          type        = "Service"
          identifiers = ["sns.amazonaws.com"]
        }
      ]

      conditions = [{
        test     = "ArnEquals"
        variable = "aws:SourceArn"
        values   = [module.incorrect_information.topic_arn]
      }]
    }
  }

  tags = {
    Environment = "dev"
  }
}
```

- order main.tf

```
## sns to sqs
## mailprosessing
module "quote-order" {
  source  = "terraform-aws-modules/sns/aws"
  version = ">= 5.0"

  name = "quote-order"
```

```
  topic_policy_statements = {
    sqs = {
      sid       = "SQSSubscribe"
      actions   = ["sns:Subscribe", "sns:Receive"]
      principals = [{
        type        = "AWS"
        identifiers = ["*"]
      }]
      conditions = [{
        test     = "StringLike"
        variable = "sns:Endpoint"
        values   = [module.quote_order_sqs.queue_arn, module.
      }]
    }
  }

  subscriptions = {
    quote_order_sqs = {
      protocol = "sqs"
      endpoint = module.quote_order_sqs.queue_arn
    }
    quote_order_save_sqs = {
      protocol = "sqs"
      endpoint = module.quote_order_save_sqs.queue_arn
    }
  }

  tags = {
    Environment = "dev"
  }
}


##quote-order-trigger

module "quote_order_sqs" {
  source = "terraform-aws-modules/sqs/aws"
```

```
  name = "quote-order-trigger"

  create_queue_policy = true
  queue_policy_statements = {
    sns = {
      sid     = "SNSPublish"
      actions = ["sqs:SendMessage"]

      principals = [
        {
          type        = "Service"
          identifiers = ["sns.amazonaws.com"]
        }
      ]

      conditions = [{
        test     = "ArnEquals"
        variable = "aws:SourceArn"
        values   = [module.quote-order.topic_arn]
      }]
    }
  }

  tags = {
    Environment = "dev"
  }
}

##quote-order-save-trigger

module "quote_order_save_sqs" {
  source = "terraform-aws-modules/sqs/aws"

  name = "quote-order-save-trigger"

  create_queue_policy = true
  queue_policy_statements = {
    sns = {
```

```
      sid     = "SNSPublish"
      actions = ["sqs:SendMessage"]

      principals = [
        {
          type        = "Service"
          identifiers = ["sns.amazonaws.com"]
        }
      ]

      conditions = [{
        test     = "ArnEquals"
        variable = "aws:SourceArn"
        values   = [module.quote-order.topic_arn]
      }]
    }
  }

  tags = {
    Environment = "dev"
  }
}
```

- request-mail main.tf

```
## SQS Queues
resource "aws_sqs_queue" "file_classification_trigger" {
  name = "file-classification-trigger"
}

resource "aws_sqs_queue" "mail_classification_trigger" {
  name = "mail-classification-trigger"
}

resource "aws_sqs_queue" "unzip_trigger" {
  name = "unzip-trigger"
}
```

```
resource "aws_sqs_queue" "file_decoding_trigger" {
  name = "file-decoding-trigger"
}

resource "aws_sqs_queue" "file_mail_classification_trigger" {
  name = "file-mail-classification-trigger"
}

resource "aws_sqs_queue" "zip_mail_classification_trigger" {
  name = "zip-mail-classification-trigger"
}


## S3 Bucket
resource "aws_s3_bucket" "request_mail" {
  bucket         = "request-mail"
  force_destroy  = true
  tags = {
    environment = "devel"
  }
}

resource "aws_s3_bucket" "mails_to_files" {
  bucket         = "mails-to-files"
  force_destroy  = true
  tags = {
    environment = "devel"
  }
}


## ECR Repository
resource "aws_ecr_repository" "mail_classfication" {
  name                 = "mail-classfication"
  image_tag_mutability = "MUTABLE"
}
```

```
## S3 Bucket Public Access Block
resource "aws_s3_bucket_public_access_block" "public_access_b
  bucket = aws_s3_bucket.request_mail.id

  block_public_acls       = false
  block_public_policy     = false
  ignore_public_acls      = false
  restrict_public_buckets = false
}

## S3 Bucket Policy
resource "aws_s3_bucket_policy" "bucket_policy" {
  bucket = aws_s3_bucket.request_mail.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid       = "AllowSESToPutEmails",
        Effect    = "Allow",
        Principal = {
          Service = "ses.amazonaws.com"
        },
        Action    = [
          "s3:PutObject",
          "s3:PutObjectAcl",
        ],
        Resource  = "${aws_s3_bucket.request_mail.arn}/*"
      },
      {
        Sid       = "PublicRead",
        Effect    = "Allow",
        Principal = "*",
        Action    = "s3:GetObject",
        Resource  = "${aws_s3_bucket.request_mail.arn}/*"
      },
    ]
  })
```

```
}

## S3 Bucket Notification for SQS
resource "aws_s3_bucket_notification" "mail_classfication_tri
  bucket = aws_s3_bucket.request_mail.id

  queue {
    queue_arn = aws_sqs_queue.file_classification_trigger.arn
    events    = ["s3:ObjectCreated:*"]
  }

  depends_on = [aws_sqs_queue_policy.s3_to_sqs_policy]
}

## SQS Queue Policy for S3 to Send Messages to SQS Queue
resource "aws_sqs_queue_policy" "s3_to_sqs_policy" {
  queue_url = aws_sqs_queue.file_classification_trigger.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow",
        Principal = {
          Service = "s3.amazonaws.com"
        },
        Action = "SQS:SendMessage",
        Resource = aws_sqs_queue.file_classification_trigger.
        Condition = {
          ArnLike = {
            "aws:SourceArn" = aws_s3_bucket.request_mail.arn
          }
        }
      }
    ]
  })
}
```

- route53 main.tf

```
# S3 버킷 생성 - 공용 접근 제거
resource "aws_s3_bucket" "front_bucket" {
  bucket = "modomail-bucket"
  website {
    index_document = "index.html"
    error_document = "index.html"
  }

  tags = {
    Name = "ReactAppBucket"
  }
}


# CloudFront Origin Access Identity
resource "aws_cloudfront_origin_access_identity" "origin_iden
  comment = "Access identity for React app S3 bucket"
}

# CloudFront 배포 생성
resource "aws_cloudfront_distribution" "front_distribution" {
  origin {
    domain_name = aws_s3_bucket.front_bucket.bucket_regional_
    origin_id   = "S3-origin-react-app"

    s3_origin_config {
      origin_access_identity = aws_cloudfront_origin_access_i
    }
  }

  enabled             = true
  is_ipv6_enabled     = true
  default_root_object = "index.html"

  default_cache_behavior {
```

```
    allowed_methods  = ["GET", "HEAD", "OPTIONS"]
    cached_methods   = ["GET", "HEAD"]
    target_origin_id = "S3-origin-react-app"

    forwarded_values {
      query_string = false
      cookies {
        forward = "none"
      }
    }

    viewer_protocol_policy = "redirect-to-https"
    min_ttl                = 0
    default_ttl            = 3600
    max_ttl                = 86400
  }

  price_class = "PriceClass_100"

  # 대체 도메인 이름 설정
  aliases = ["busybeemail.net"]  # 대체 도메인 이름을 설정

  # SSL 인증서 설정 (ACM에서 발급받은 인증서 ARN으로 변경)
  viewer_certificate {
    acm_certificate_arn = "arn:aws:acm:us-east-1:481665114066
    ssl_support_method  = "sni-only"
  }

  restrictions {
    geo_restriction {
      restriction_type = "none"
    }
  }

  tags = {
    Name = "ReactAppDistribution"
  }
}
```

```
# S3 버킷 정책 - CloudFront OAI 접근만 허용
resource "aws_s3_bucket_policy" "react_app_bucket_policy" {
  bucket = aws_s3_bucket.front_bucket.id

  policy = jsonencode({
    Version = "2012-10-17",
    Id      = "http referer policy example",
    Statement = [
      {
        Sid       = "Stmt1730795456871",
        Effect    = "Allow",
        Principal = "*",
        Action    = ["s3:GetObject"],
        Resource  = "arn:aws:s3:::modomail-bucket/*"
      }
    ]
  })
}


# Route53 레코드 설정
resource "aws_route53_record" "front_record" {
  zone_id = aws_route53_zone.my_zone.zone_id
  name    = "www.busybeemail.net"  # 본인의 서브도메인으로 변경
  type    = "A"

  alias {
    name                   = aws_cloudfront_distribution.fron
    zone_id                = aws_cloudfront_distribution.fron
    evaluate_target_health = false
  }
}

resource "aws_route53_zone" "my_zone" {
  name = "busybeemail.net"
}
```

```
resource "aws_route53_record" "mail" {
  zone_id = aws_route53_zone.my_zone.zone_id
  name    = "busybeemail.net"
  type    = "MX"
  ttl     = 300
  records = ["10 inbound-smtp.ap-northeast-2.amazonaws.com"]
}

resource "aws_route53_record" "dmarc" {
  zone_id = aws_route53_zone.my_zone.zone_id
  name    = "_dmarc.busybeemail.net"
  type    = "TXT"
  ttl     = 300
  records = ["v=DMARC1; p=none;"]
}

resource "aws_route53_record" "spf" {
  zone_id = aws_route53_zone.my_zone.zone_id
  name    = "busybeemail.net"
  type    = "TXT"
  ttl     = 300
  records = ["v=spf1 include:amazonses.com ~all"]
}
output "route53_zone_id" {
  value = aws_route53_zone.my_zone.zone_id
}
```

- ses maint.tf

```
resource "aws_ses_domain_identity" "domain_identity" {
  domain = "busybeemail.net"  # 본인의 도메인으로 변경
}


resource "aws_ses_receipt_rule_set" "rule_set" {
```

```
  rule_set_name = "mail_rule_set"
}

resource "aws_ses_receipt_rule" "email_rule" {
  rule_set_name = aws_ses_receipt_rule_set.rule_set.rule_set_
  name          = "s3_mail_rule"
  enabled       = true
  recipients    = []  # 수신할 이메일 주소 또는 도메인 지정

  s3_action {
      bucket_name = "request-mail"  # 실제 S3 버킷 이름으로 변경
      object_key_prefix = "mails/"  # 저장할 폴더 경로 (선택 사항)
      kms_key_arn = null             # KMS 키 ARN 설정 (암호화가
      position    = 1
      topic_arn   = null             # 알림이 필요할 경우 SNS 주제
   }

  # 수신 거부 규칙 (선택 사항)
  scan_enabled = true
  tls_policy   = "Optional"
}

resource "aws_iam_policy" "ses_s3" {
  name        = "ses"
  description = "Policy to allow SES to save emails to S3"

  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Action = [
          "s3:*",
          "s3-object-lambda:*",
          "s3:PutObject",
          "s3:PutObjectAcl"
        ],
        Effect   = "Allow",
        Resource = "arn:aws:s3:::request-mail/*"   # 실제 S3 버
```

```
      }
    ]
  })
}

resource "aws_iam_role" "ses_s3_roles" {
  name = "ses_s3_roles"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action    = "sts:AssumeRole"
        Principal = {
          Service = "ses.amazonaws.com"
        }
        Effect    = "Allow"
        Sid       = ""
      },
    ]
  })
}


resource "aws_iam_role_policy_attachment" "ses_s3_attachment"
  policy_arn = aws_iam_policy.ses_s3.arn
  role       = aws_iam_role.ses_s3_roles.name
}

# ## route53에 cname 등록
# module "route53_zone" {
#   source = "../route53"  # 실제 경로로 변경
# }


# resource "aws_route53_record" "ses_verification" {
#   zone_id = module.route53_zone.zone_id
#   name    = "${aws_ses_domain_identity.domain_identity.doma
```

```
#   type    = "CNAME"
#   ttl     = 300

#   records = [
#     "CNAME_record_value_from_SES"  # SES에서 제공하는 Value 값
#   ]
# }
```

- ssm main.tf

```
# SSM Parameter 생성
resource "aws_ssm_parameter" "openai_api_key" {
  name  = "/prod/openAI/api_key"
  type  = "SecureString"
  value = "sk-proj-7I5qg6q-ETorcXOtKMIAbtOh1orsMEB-AXlt1oEGQp
  description = "OpenAI API Key for production environment"
  overwrite   = true
}

resource "aws_ssm_parameter" "vpc_subnet" {
  name = "/network/private-subnet-id"
  type = "String"
  value = "subnet-013373120d515f276"
  description = "vpc-subnet-id"
  overwrite = true
}

resource "aws_ssm_parameter" "vpc_security_group_subnet" {
  name = "/network/lambda-security-group-id"
  type = "String"
  value = "sg-0076703612c9973df"
  description = "vpc-security-group-id"
  overwrite = true
}

resource "aws_ssm_parameter" "tavily_api_key" {
  name = "/prod/tavily/api_key"
  type  = "SecureString"
```

```
  value = "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"
  description = "tavily-api-key"
  overwrite   = true
}
```

- vpc main.tf

```
# VPC 생성
resource "aws_vpc" "vpc" {
  cidr_block = "10.0.0.0/16"
  enable_dns_support   = true
  enable_dns_hostnames = true
  tags = {
    Name = "vpc"
  }
}

# 인터넷 게이트웨이 생성
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.vpc.id
  tags = {
    Name = "igw"
  }
}

# 퍼블릭 서브넷 생성
resource "aws_subnet" "public_subnet" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = "10.0.1.0/24"

  availability_zone = "ap-northeast-2a" # 원하는 가용 영역으로 변경
  map_public_ip_on_launch = true
  tags = {
    Name = "public-subnet"
  }
}
```

```
# 프라이빗 서브넷 생성
resource "aws_subnet" "private_subnet" {
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = "10.0.2.0/24"
  availability_zone = "ap-northeast-2a"  # 원하는 가용 영역으로 변
  tags = {
    Name = "private-subnet"
  }
}

# 퍼블릭 라우팅 테이블 생성
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }
  tags = {
    Name = "public-rt"
  }
}

# 퍼블릭 서브넷과 라우팅 테이블 연결
resource "aws_route_table_association" "public_rta" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.public_rt.id
}


# 퍼블릭 서브넷용 보안 그룹
resource "aws_security_group" "public_sg" {
  vpc_id = aws_vpc.vpc.id
  tags = {
    Name = "public-sg"
  }

  # 인바운드 규칙 (예: HTTP 및 HTTPS 트래픽 허용)
```

```
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # 모든 IP로부터 HTTP 허용
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # 모든 IP로부터 HTTPS 허용
  }

  # 아웃바운드 규칙 (모든 아웃바운드 트래픽 허용)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# 프라이빗 서브넷용 보안 그룹
resource "aws_security_group" "private_sg" {
  vpc_id = aws_vpc.vpc.id
  tags = {
    Name = "private-sg"
  }

  # 인바운드 규칙 (예: 퍼블릭 서브넷의 리소스에서 오는 트래픽만 허용)
  ingress {
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [aws_security_group.public_sg.id]  # 퍼블
  }
```

```
  # 아웃바운드 규칙 (모든 아웃바운드 트래픽 허용)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}


# 출력 변수 설정 (VPC와 서브넷 ID 출력)
output "vpc_id" {
  value = aws_vpc.vpc.id
}

output "public_subnet_id" {
  value = aws_subnet.public_subnet.id
}

output "private_subnet_id" {
  value = aws_subnet.private_subnet.id
}
```

- main.tf

```
module "sqs_module" {
  source = "./modules/request-mail"
}

module "classfication_module" {
    source = "./modules/classification"
}

module "information_verification" {
  source = "./modules/information_verification"
}
```

```
module "chat" {
  source = "./modules/chat"
}

module "vpc" {
  source = "./modules/vpc"
}

module "ses" {
  source = "./modules/ses"
}

module "route53" {
  source = "./modules/route53"
}

module "cognito_pool" {
  source = "./modules/coginito"
}

module "ssm" {
  source = "./modules/ssm"
}

module "order" {
  source = "./modules/order"
}
```

- provider.tf

```
module "sqs_module" {
  source = "./modules/request-mail"
}

module "classfication_module" {
    source = "./modules/classification"
```

```
}

module "information_verification" {
  source = "./modules/information_verification"
}

module "chat" {
  source = "./modules/chat"
}

module "vpc" {
  source = "./modules/vpc"
}

module "ses" {
  source = "./modules/ses"
}

module "route53" {
  source = "./modules/route53"
}

module "cognito_pool" {
  source = "./modules/coginito"
}

module "ssm" {
  source = "./modules/ssm"
}

module "order" {
  source = "./modules/order"
}
```

terraform init → terraform plan → terraform apply

5. jenkins

- file-classification

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-file-classification',
```

```
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('install Serverless Framework and deploy') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            npm install -g serverless || exit
                            cd functions/file-classification
                            export AWS_ACCESS_KEY_ID=$AWS_ACC
                            export AWS_SECRET_ACCESS_KEY=$AWS
                            export SERVERLESS_ACCESS_KEY=$SER
                            sls deploy --stage dev
                        '''
                    }
                }
            }
            post {
                success {
                    echo 'Serverless deployment successful'
                }
                failure {
                    error 'Serverless deployment failed'
                }
            }
        }
    }
}
```

- frontend

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-frontend',
                    credentialsId: 'myt'
                sh "ls -al"
```

```
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('install Amplify CLI') {
            steps {
                script {
                    sh '''
                        npm install -g @aws-amplify/cli
                    '''
                }
            }
            post {
                success {
                    echo 'Amplify CLI installed successfully'
                }
                failure {
                    error 'Amplify CLI installation failed'
                }
            }
        }
        stage('install dependencies') {
            steps {
                script {
                    sh '''
                        cd frontend
                        npm install
                    '''
                }
            }
            post {
                success {
```

```
                    echo 'Dependencies installed successfully
                }
                failure {
                    error 'Dependency installation failed'
                }
            }
        }
        stage('build project') {
            steps {
                script {
                    sh '''
                        cd frontend
                        npm run build
                    '''
                }
            }
            post {
                success {
                    echo 'Project built successfully'
                }
                failure {
                    error 'Project build failed'
                }
            }
        }
        stage('deploy to Amplify') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            cd frontend
                            amplify publish --yes
                        '''
                    }
                }
            }
            post {
                success {
```

```
                    echo 'Deployment to Amplify successful'
                }
                failure {
                    error 'Deployment to Amplify failed'
                }
            }
        }
    }
}
```

- informatino-integrity-verification

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
```

```
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-information-integrity-ve
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('build project') {
            steps {
                script {
                    sh '''
                        cd functions/information-integrity-ve
                        echo Checking for gradlew file
                        ls
                        chmod +x gradlew
                        if [ ! -f gradlew ]; then
                            echo "Gradle Wrapper not found, b
                            exit 1
                        fi
                        ./gradlew clean shadowJar
                    '''
                }
            }
            post {
                success {
```

```
                    echo 'Project built successfully'
                }
                failure {
                    error 'Project build failed'
                }
            }
        }
        stage('install Serverless Framework and deploy') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            npm install -g serverless || exit
                            cd functions/information-integrit
                            export AWS_ACCESS_KEY_ID=$AWS_ACC
                            export AWS_SECRET_ACCESS_KEY=$AWS
                            export SERVERLESS_ACCESS_KEY=$SER
                            sls deploy --stage dev
                        '''
                    }
                }
            }
            post {
                success {
                    echo 'Serverless deployment successful'
                }
                failure {
                    error 'Serverless deployment failed'
                }
            }
        }
    }
}
```

- information-mail-request

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-information-mail-request
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
```

```
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('build project') {
            steps {
                script {
                    sh '''
                        cd functions/information-mail-request
                        echo Checking for gradlew file
                        ls
                        chmod +x gradlew
                        if [ ! -f gradlew ]; then
                            echo "Gradle Wrapper not found, b
                            exit 1
                        fi
                        ./gradlew clean shadowJar
                    '''
                }
            }
            post {
                success {
                    echo 'Project built successfully'
                }
                failure {
                    error 'Project build failed'
                }
            }
        }
        stage('install Serverless Framework and deploy') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            npm install -g serverless || exit
```

```
                            cd functions/information-mail-req
                            export AWS_ACCESS_KEY_ID=$AWS_ACC
                            export AWS_SECRET_ACCESS_KEY=$AWS_
                            export SERVERLESS_ACCESS_KEY=$SERV
                            sls deploy --stage dev
                        '''
                    }
                }
            }
            post {
                success {
                    echo 'Serverless deployment successful'
                }
                failure {
                    error 'Serverless deployment failed'
                }
            }
        }
    }
}
```

- llm-interaction

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
```

```
    stages {
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-llm-interaction',
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('install Serverless Framework and deploy') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            npm install -g serverless || exit
                            cd functions/llm-interaction
```

```
                        export AWS_ACCESS_KEY_ID=$AWS_ACC
                        export AWS_SECRET_ACCESS_KEY=$AWS
                        export SERVERLESS_ACCESS_KEY=$SER
                        sls deploy --stage dev
                    '''
                }
            }
        }
        post {
            success {
                echo 'Serverless deployment successful'
            }
            failure {
                error 'Serverless deployment failed'
            }
        }
    }
}
```

- mail-classificatino

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
```

```
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-llm-interaction',
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('install Serverless Framework and deploy') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            npm install -g serverless || exit
                            cd functions/llm-interaction
                            export AWS_ACCESS_KEY_ID=$AWS_ACC
```

```
                        export AWS_SECRET_ACCESS_KEY=$AWS
                        export SERVERLESS_ACCESS_KEY=$SER
                        sls deploy --stage dev
                    '''
                }
            }
        }
        post {
            success {
                echo 'Serverless deployment successful'
            }
            failure {
                error 'Serverless deployment failed'
            }
        }
    }
}
```

- mail-extraction

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
```

```
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-llm-interaction',
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('install Serverless Framework and deploy') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            npm install -g serverless || exit
                            cd functions/llm-interaction
                            export AWS_ACCESS_KEY_ID=$AWS_ACC
```

```
                        export AWS_SECRET_ACCESS_KEY=$AWS
                        export SERVERLESS_ACCESS_KEY=$SER
                        sls deploy --stage dev
                    '''
                }
            }
        }
        post {
            success {
                echo 'Serverless deployment successful'
            }
            failure {
                error 'Serverless deployment failed'
            }
        }
    }
}
```

- quotation-calculation

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
        stage('init') {
```

```
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-quotation-calculation',
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('build project') {
            steps {
                script {
                    sh '''
                        cd functions/quotation-calculation
                        echo Checking for gradlew file
                        ls
                        chmod +x gradlew
                        if [ ! -f gradlew ]; then
```

```
                        echo "Gradle Wrapper not found, b
                        exit 1
                    fi
                    ./gradlew clean shadowJar
                '''
            }
        }
        post {
            success {
                echo 'Project built successfully'
            }
            failure {
                error 'Project build failed'
            }
        }
    }
    stage('install Serverless Framework and deploy') {
        steps {
            withCredentials([aws(credentialsId: 'awss', a
                script {
                    sh '''
                        npm install -g serverless || exit
                        cd functions/quotation-calculatio
                        export AWS_ACCESS_KEY_ID=$AWS_ACC
                        export AWS_SECRET_ACCESS_KEY=$AWS
                        export SERVERLESS_ACCESS_KEY=$SER
                        sls deploy --stage dev
                    '''
                }
            }
        }
        post {
            success {
                echo 'Serverless deployment successful'
            }
            failure {
                error 'Serverless deployment failed'
            }
```

```
            }
        }
    }
}
```

- save-data

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
```

```
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-save-data',
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('build project') {
            steps {
                script {
                    sh '''
                        cd functions/save-data
                        echo Checking for gradlew file
                        ls
                        chmod +x gradlew
                        if [ ! -f gradlew ]; then
                            echo "Gradle Wrapper not found, b
                            exit 1
                        fi
                        ./gradlew clean shadowJar
                    '''
                }
            }
            post {
                success {
                    echo 'Project built successfully'
                }
                failure {
                    error 'Project build failed'
                }
```

```
            }
        }
        stage('install Serverless Framework and deploy') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            npm install -g serverless || exit
                            cd functions/save-data
                            export AWS_ACCESS_KEY_ID=$AWS_ACC
                            export AWS_SECRET_ACCESS_KEY=$AWS_
                            export SERVERLESS_ACCESS_KEY=$SER
                            sls deploy --stage dev
                        '''
                    }
                }
            }
            post {
                success {
                    echo 'Serverless deployment successful'
                }
                failure {
                    error 'Serverless deployment failed'
                }
            }
        }
    }
}
```

- send-quote-mail

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
```

```
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-send-quote-mail',
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
```

```
        }
        stage('build project') {
            steps {
                script {
                    sh '''
                        cd functions/send-quote-mail
                        echo Checking for gradlew file
                        ls
                        chmod +x gradlew
                        if [ ! -f gradlew ]; then
                            echo "Gradle Wrapper not found, b
                            exit 1
                        fi
                        ./gradlew clean shadowJar
                    '''
                }
            }
            post {
                success {
                    echo 'Project built successfully'
                }
                failure {
                    error 'Project build failed'
                }
            }
        }
        stage('install Serverless Framework and deploy') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            npm install -g serverless || exit
                            cd functions/send-quote-mail
                            export AWS_ACCESS_KEY_ID=$AWS_ACC
                            export AWS_SECRET_ACCESS_KEY=$AWS
                            export SERVERLESS_ACCESS_KEY=$SER
                            sls deploy --stage dev
                        '''
```

```
                    }
                }
            }
            post {
                success {
                    echo 'Serverless deployment successful'
                }
                failure {
                    error 'Serverless deployment failed'
                }
            }
        }
    }
}
```

- zip-classification

```
pipeline {
    agent any
    options {
        timeout(time: 1, unit: 'HOURS') // set timeout to 1 h
    }
    tools {
        nodejs "node" // Jenkins에 등록된 Node.js 설정
    }
    environment {
        AWS_ACCOUNT_ID = '047719649915'
        REGION = 'ap-northeast-2'
        AWS_CREDENTIAL_NAME = 'awss' // Jenkins에 등록한 AWS Cr
        SERVERLESS_ACCESS_KEY = credentials('SERVERLESS_ACCES
    }
    stages {
        stage('init') {
            steps {
                echo 'Initializing workspace...'
                deleteDir()
            }
```

```
            post {
                success {
                    echo 'Initialization successful'
                }
                failure {
                    error 'Initialization failed'
                }
            }
        }
        stage('clone project') {
            steps {
                git url: 'https://lab.ssafy.com/s11-final/S11
                    branch: 'develop-zip-classification',
                    credentialsId: 'myt'
                sh "ls -al"
            }
            post {
                success {
                    echo 'Project cloned successfully'
                }
                failure {
                    error 'Project clone failed'
                }
            }
        }
        stage('install Serverless Framework and deploy') {
            steps {
                withCredentials([aws(credentialsId: 'awss', a
                    script {
                        sh '''
                            npm install -g serverless || exit
                            cd functions/zip-classification
                            export AWS_ACCESS_KEY_ID=$AWS_ACC
                            export AWS_SECRET_ACCESS_KEY=$AWS
                            export SERVERLESS_ACCESS_KEY=$SER
                            sls deploy --stage dev
                        '''
                    }
```

```
                }
            }
            post {
                success {
                    echo 'Serverless deployment successful'
                }
                failure {
                    error 'Serverless deployment failed'
                }
            }
        }
    }
}
```

6. 아두이노

- ec2 인스턴스 생성
- alb 를 통한 로드벨런싱
- alb등록법 대상 등록 alb 대상 https 443 http 80 으로 로드벨런싱 후 route53 에 등록

ec2 접근하여 mqtt 설치

```
sudo apt update
sudo apt install mosquitto mosquitto-clients
sudo systemctl status mosquitto
sudo systemctl enable mosquitto
```

node 설치

```
curl -sL https://deb.nodesource.com/setup_lts.x | sudo -E bas
sudo apt install nodejs
```

r4 socket 브로커

```
const mqtt = require('mqtt');
const WebSocket = require('ws');
const express = require('express');


// MQTT 브로커 URL
const brokerUrl = "ws://52.78.59.81:8080"; // MQTT 브로커 WebSo

// MQTT 클라이언트 생성
const mqttClient = mqtt.connect(brokerUrl);

// WebSocket 서버 생성
const wss = new WebSocket.Server({ port: 8082 }); // WebSocke

// 연결된 WebSocket 클라이언트 목록
const connectedClients = new Set();

// WebSocket 클라이언트 연결 이벤트
wss.on('connection', (ws) => {
    console.log('WebSocket 클라이언트가 연결되었습니다.');
    connectedClients.add(ws);

    // WebSocket 연결 해제 이벤트
    ws.on('close', () => {
        console.log('WebSocket 클라이언트가 연결 해제되었습니다.');
        connectedClients.delete(ws);
    });

    // WebSocket 클라이언트 메시지 처리 (옵션)
    ws.on('message', (message) => {
        console.log('WebSocket 클라이언트로부터 메시지 수신:', mess
```

```
    });

    // WebSocket 에러 처리
    ws.on('error', (err) => {
        console.error('WebSocket 클라이언트 에러:', err);
    });
});

// MQTT 브로커 연결 이벤트
mqttClient.on('connect', () => {
    console.log('MQTT 클라이언트가 브로커에 연결되었습니다.');

    // 특정 토픽 구독
    const topic = ['sensor/data', 'gps/data'];
    mqttClient.subscribe(topic, (err) => {
        if (err) {
            console.error('토픽 구독 실패:', err);
        } else {
            console.log(`"${topic}" 토픽 구독 성공.`);
        }
    });
});


// MQTT 메시지 수신 이벤트
mqttClient.on('message', (topic, message) => {
    console.log(`MQTT "${topic}" 토픽에서 메시지를 수신했습니다: ${m

    // 수신된 메시지 JSON 파싱
    const parsedMessage = JSON.parse(message.toString());

    // sensor/data 또는 gps/data 토픽 처리
    if (topic === 'sensor/data' || topic === 'gps/data') {
        connectedClients.forEach((ws) => {
            if (ws.readyState === WebSocket.OPEN) {
                try {
                    // 연결된 wss 클라이언트로 토픽과 데이터 브로드캐스
                    ws.send(JSON.stringify({ topic, data: par
```

```
                } catch (err) {
                    console.error('WebSocket 메시지 전송 중 에러:
                }
            }
        });
    }
});

// MQTT 오류 이벤트
mqttClient.on('error', (err) => {
    console.error('MQTT 클라이언트 오류:', err);
});

// WebSocket 서버 종료 처리
wss.on('close', () => {
    console.log('WebSocket 서버가 종료되었습니다.');
});

// MQTT 클라이언트 종료 처리
mqttClient.on('close', () => {
    console.log('MQTT 클라이언트 연결이 종료되었습니다.');
});

// WebSocket 서버 실행
console.log('WebSocket 서버가 8082 포트에서 실행 중입니다.');
```

webcam 브로커

```
const fs = require('fs');
const path = require('path');
const WebSocket = require('ws');
const express = require('express');

// Express 서버 설정
const app = express();
const PORT = 8081;
```

```
// 정적 파일 제공
app.use(express.static(path.join(__dirname, 'public')));

// WebSocket 서버 설정
const wss = new WebSocket.Server({ noServer: true });

// WebSocket 브로드캐스트 함수
const broadcastImage = (data) => {
  // base64로 변환하여 전송
  const base64Data = Buffer.from(data).toString('base64');
  wss.clients.forEach((client) => {
    if (client.readyState === WebSocket.OPEN) {
      client.send(base64Data); // 모든 연결된 클라이언트로 이미지 데
    }
  });
};

// WebSocket 연결 이벤트
wss.on('connection', (ws) => {
  console.log('New WebSocket client connected.');

  // 클라이언트 연결 종료 이벤트
  ws.on('close', () => {
    console.log('WebSocket client disconnected.');
  });
});

// HTTP 서버와 WebSocket 통합
const server = app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`
});
server.on('upgrade', (request, socket, head) => {
  wss.handleUpgrade(request, socket, head, (ws) => {
    wss.emit('connection', ws, request);
  });
});

// WebSocket 메시지 처리
```

```
wss.on('connection', (ws) => {
  ws.on('message', (data) => {
    console.log(`Received data of size: ${data.length} bytes`

    // 이미지 데이터를 브로드캐스트
    broadcastImage(data);
  });
});
```

r4 wifi , 온도습도센서, gps센서 , rfid 센서, 초음파 센서

```
#include "secret.h"            // WiFi 및 MQTT 서버 정보가 저장된
#include <WiFiS3.h>           // WiFiS3 라이브러리 포함
#include <PubSubClient.h>     // PubSubClient 라이브러리 포함
#include <DHT.h>              // DHT 라이브러리 포함
#include <MFRC522.h>          // RFID 라이브러리 포함
#include <ArduinoJson.h>      // ArduinoJson 라이브러리 포함

#define TRIG 6          // TRIG 핀 설정 (초음파 보내는 핀)
#define ECHO 7          // ECHO 핀 설정 (초음파 받는 핀)
#define THRESHOLD 40    // 감지 거리 임계값 (cm)


const char* ssid = "ssid";             // WiFi SSID
const char* password = "password";     // WiFi 비밀번호

// MQTT 서버 정보
const char* mqtt_server = "ip"; // Mosquitto 브로커 EC2 IP
// SensorData 구조체 선언
struct SensorData {

  float temperature;   // 온도
  float humidity;      // 습도
  bool isOpen;        // 문 열림 상태 (문이 열렸으면 true, 아니면 fa
  int status = 3;      // 상태 정보 (3, 4, 5)
};
```

```
// 전역 구조체 변수 선언
SensorData sensorData;
WiFiClient espClient;
PubSubClient client(espClient);

// DHT 센서 설정
#define DHTPIN 4          // DHT 센서를 연결한 핀 번호
#define DHTTYPE DHT11     // DHT 센서 종류 (DHT11)
DHT dht(DHTPIN, DHTTYPE); // DHT 센서 객체 생성

// RFID 설정
#define SS_PIN 10               // RFID SS 핀 (SPI 핀: 10번)
#define RST_PIN 9               // RFID RST 핀 (SPI 핀: 9번)
MFRC522 rfid(SS_PIN, RST_PIN);  // RFID 객체 생성

void setup() {
  Serial.begin(9600);

  // WiFi 연결 설정
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  } if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Failed to connect to WiFi. Please check c
  } else {
    Serial.println("Connected to WiFi");
  }

  // MQTT 설정
  client.setServer(mqtt_server, 1883);
  client.setCallback(mqttCallback);  // MQTT 콜백 함수 등록

  // 센서 설정
  setupSensors();
}
```

```
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  // 센서 데이터를 수집하여 JSON으로 변환
  collectSensorData();
  String jsonData = convertToJSON(sensorData);
  Serial.println("jsonData = " + jsonData);

  char jsonBuffer[256];
  jsonData.toCharArray(jsonBuffer, jsonData.length() + 1);

  // MQTT 주제에 JSON 데이터 발행
  client.publish("sensor/data", jsonBuffer);

  delay(2500);  // 2.5초마다 데이터 발행
}

// 센서 초기화 함수
void setupSensors() {
  // DHT 센서 시작
  dht.begin();

  // 초음파 센서 시작
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);

  // RFID SPI 통신 시작
  SPI.begin();

  // RFID 모듈 초기화
  rfid.PCD_Init();

  Serial.println("Sensors initialized.");
}
// MQTT 재연결 함수
```

```
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ArduinoClient")) {
      Serial.println("connected");
      // 특정 주제 구독
      client.subscribe("sensor/control");  // "sensor/control
      Serial.println("Subscribed to topic: sensor/control");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      delay(2500);  // 2.5초 대기 후 재시도
    }
  }
}


// 센서 데이터 수집 함수
void collectSensorData() {
    // 온도 및 습도 데이터 읽기
  sensorData.humidity = dht.readHumidity();
  sensorData.temperature = dht.readTemperature();

  if (isnan(sensorData.humidity) || isnan(sensorData.temperat
    Serial.println("Failed to read from DHT sensor! Retrying.
    delay(1000);
    collectSensorData();  // 재시도
  }

  // RFID 카드 읽기
  if (rfid.PICC_IsNewCardPresent() && rfid.PICC_ReadCardSeria
    String cardID = getRFIDCardID(rfid);  // RFID 카드 ID 추출
    Serial.print("RFID 카드 ID: ");
    Serial.println(cardID); // 카드 ID 출력

    cardID.trim();
    cardID.toUpperCase();
```

```
    if (cardID.equals("02 20 CE 01")) { // 파란색 태그
      sensorData.status = 4;
    } else if (cardID.equals("95 00 3A 02")) { // 흰색 카드
      sensorData.status = 5;
    }

    rfid.PICC_HaltA();
  }

  // 초음파 센서
  int distance = measureDistance(); // 거리 측정
  Serial.print("측정 거리: ");
  Serial.println(distance);

  if (distance >= THRESHOLD) {
    sensorData.isOpen = true;
  } else {
    sensorData.isOpen = false;
  }
}

// RFID 카드 ID를 추출하여 String으로 반환하는 함수
String getRFIDCardID(MFRC522 &rfid) {
  String cardID = "";
  for (byte i = 0; i < rfid.uid.size; i++) {
    cardID += String(rfid.uid.uidByte[i] < 0x10 ? " 0" : " ");
    cardID += String(rfid.uid.uidByte[i], HEX);
  }
  cardID.trim();
  return cardID;
}

// 초음파 센서를 통해 거리 측정 및 물체 감지 여부 반환
int measureDistance() {
  long duration, distance;

  digitalWrite(TRIG, LOW);
  delayMicroseconds(2);
```

```
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  duration = pulseIn(ECHO, HIGH);   // ECHO 핀으로 신호가 돌아오는
  distance = duration * 17 / 1000;  // 시간을 거리로 변환 (cm 단위

  return distance;
}

// SensorData 구조체 -> JSON변환함수
String convertToJSON(SensorData data) {
  StaticJsonDocument<256> jsonDoc;

  // JSON 객체에 구조체 데이터 추가
  jsonDoc["isOpen"] = data.isOpen;
  jsonDoc["temperature"] = data.temperature;
  jsonDoc["humidity"] = data.humidity;
  jsonDoc["status"] = data.status;

  // JSON 문서를 문자열로 변환
  String jsonString;
  serializeJson(jsonDoc, jsonString);

  return jsonString;
}

// MQTT 메시지를 수신하는 콜백 함수
void mqttCallback(char* topic, byte* payload, unsigned int le
  Serial.print("Message arrived on topic: ");
  Serial.println(topic);

  // 수신된 메시지를 문자열로 변환
  String message;
  for (unsigned int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
  Serial.print("Message: ");
```

```
  Serial.println(message);

  // 특정 요청 처리
  if (String(topic) == "sensor/control") {
    if (message == "status_3") {
      sensorData.status = 3;
      Serial.println("Status updated to 3");
    } else if (message == "status_4") {
      sensorData.status = 4;
      Serial.println("Status updated to 4");
    } else if (message == "status_5") {
      sensorData.status = 5;
      Serial.println("Status updated to 5");
    } else {
      Serial.println("Invalid status update command");
    }
  }
}
```

카메라 센서

esp32-cam 및 보드

```
#include <BTAddress.h>
#include <BTAdvertisedDevice.h>
#include <BTScan.h>
#include <BluetoothSerial.h>

#include "esp_camera.h"
#include <WiFi.h>
#include <WebSocketsClient.h>  // WebSocket 클라이언트 라이브러리

// Wi-Fi 정보
const char *ssid = "ssid";
const char *password = "비번";
```

```
// WebSocket 서버 정보
const char* serverHost = "서버";  // Node.js 서버 IP
const uint16_t serverPort = 8081;  // WebSocket 서버 포트
const char* serverPath = "/";  // 기본 URL 경로
const char* protocol = "arduino";  // 프로토콜 이름

// WebSocket 클라이언트 객체
WebSocketsClient webSocket;

// 카메라 설정
#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

void setup() {
  Serial.begin(9600);

  // Wi-Fi 연결
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");

  // WebSocket 연결
  webSocket.begin(serverHost, serverPort, serverPath, protoco
  webSocket.onEvent(webSocketEvent);

  // 카메라 설정
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
```

```
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_CIF;
  config.pixel_format = PIXFORMAT_JPEG;
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 55;
  config.fb_count = 2;

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.println("Camera init failed");
    return;
  }
}

void webSocketEvent(WStype_t type, uint8_t * payload, size_t
  switch (type) {
    case WStype_CONNECTED:
      Serial.println("Connected to WebSocket server");
      break;
    case WStype_DISCONNECTED:
      Serial.println("Disconnected from WebSocket server");
      break;
    case WStype_ERROR:
      Serial.println("WebSocket Error");
      break;
    case WStype_PONG:
      Serial.println("Received PONG from server");
```

```
      break;
    default:
      break;
  }
}

void loop() {
  webSocket.loop();  // WebSocket 핸들링

  // 카메라에서 이미지 캡처
  camera_fb_t *fb = esp_camera_fb_get();
  if (fb) {
    Serial.printf("Image size: %d bytes\n", fb->len);

    // WebSocket 연결이 되면 이미지 데이터를 보내기
    if (webSocket.isConnected()) {
      webSocket.sendBIN(fb->buf, fb->len);  // 이미지 전송
    }

    // 프레임 버퍼 반환
    esp_camera_fb_return(fb);
  }
}
```