Design summary (conceptual)

1. **PR format** (developer must adhere):

```
 Function App - "bulksaandprocessor-qa-1"
Environment - "QA"         # or "Dev", or "Dev,QA"
```

2. **Lifecycle**
   - Developer opens PR in `Developers` repo → PR gets reviewed/approved and merged to `main`.
   - **Build pipeline** (CI) triggers on `main` (merge commit). It:
     - Reads the merge commit message (which contains PR title/body),
     - Parses `FUNCTION_NAME` and `TARGET_ENVS`,
     - Builds artifacts and publishes them,
     - Triggers the `Azure_build_pipelines` release via Azure DevOps REST API and passes variables `FUNCTION_NAME`, `TARGET_ENVS`, and the `buildId`.
     - If the build is currently running (there was an earlier build for `main`), the CI waits until build completes or it triggers the release only after successful build — we accomplish this by waiting inside CI for the build that triggered the pipeline to finish before creating the release (see details).
   - **Release** (`Azure_build_pipelines`) receives the request (release with variables). The release pipeline contains many jobs (one job per Function App). Each job has a **custom condition** that checks:
     - job runs only if `FUNCTION_NAME` equals the job's Function App name
     - and `TARGET_ENVS` includes the environment for that job.
   - The selected job executes:
     - uses Azure CLI to **start** the target Function App (look up resource group dynamically),
     - deploys artifact (zip deploy or built artifact),
     - performs health check,
     - logs results and sends notification on failure.
   - **Concurrency & Priority**:
     - Use Azure DevOps **environment** concurrency limits (set environment to allow 1 deployment at a time → queued deployments run FIFO).
     - Releases created are queued — as they're created, they will be processed FIFO by Azure DevOps environment if only one parallel deployment allowed.
   - **Notifications**:
     - On failure use Azure DevOps notification system or a pipeline step to send email (SMTP or Office365) using pipeline variables.
   - **Security**:
     - Use `System.AccessToken` for REST calls (enable OAuth access in pipeline).
     - Limit service connection scope and RBAC to only required subscriptions/resource groups.

Files & repo layout (what to add to `Developers` repo)

```
├── .azure-pipelines/
│   ├── ci-main.yml               # CI pipeline for main (triggered on merge)
│   └── trigger_release.ps1       # script to create release via REST
├── release/
│   └── azure_build_release.yml   # YAML representation (or use classic release)
├── scripts/
│   ├── deploy-function.ps1       # per-job deployment script (start, deploy, healthcheck)
│   └── utils.ps1                 # helper functions (get resourceGroup, send-email)
└── README.md
```

> You already have `Azure_build_pipelines` (classic release). Provided a YAML release for clarity — you can map logic into your classic release definition by applying same conditions and steps.

High-Level Flow

```
[PR in Developers repo]
       |
       v
[Extract FunctionApp + Env from PR description]
       |
       v
[On merge -> CI Pipeline runs]
       |
       v
[Check if Azure_build_pipelines build is running]
       | (wait until finished)
       v
[Enable matching agent in release pipeline]
       |
       v
[Enable Azure Function App in target env(s)]
       |
       v
[Trigger release for Azure_build_pipelines with correct stage(s)]
       |
       v
[Email notification on success/failure]
```

Detailed pipelines & scripts (copy/paste ready)

> **Important before using**:
>
> - Replace placeholders (like `<YOUR_AZURE_SERVICE_CONNECTION>`, `<RELEASE_DEFINITION_ID>`, `<SMTP_SERVER>`, emails, etc).
> - In pipeline settings: **Enable** "Allow scripts to access the OAuth token" for any job that uses `$(System.AccessToken)`.
> - Ensure your service connection has permissions to manage Function Apps.
> - Ensure `az cli` is available on agents or use `AzureCLI@2` task.

---

1) CI pipeline — `.azure-pipelines/ci-main.yml`

This pipeline runs **when changes are merged to main**. It:

- Parses the merge commit message for `Function App` and `Environment`
- Builds (placeholder) — replace with real build tasks for your projects
- Publishes artifact
- Triggers `Azure_build_pipelines` release via REST, passing `FUNCTION_NAME`, `TARGET_ENVS` and `buildId`.

```yaml
# .azure-pipelines/ci-main.yml
trigger:
  branches:
    include:
      - main

pool:
  vmImage: 'ubuntu-latest'

variables:
  azureServiceConnection: '<YOUR_AZURE_SERVICE_CONNECTION>'   # not for REST, for az tasks if used
  releaseDefinitionId: '<RELEASE_DEF_ID>'                     # ID of Azure_build_pipelines release definition
  releaseProject: '$(System.TeamProject)'

steps:
- checkout: self
  persistCredentials: true

# 1) Parse the merge commit message for FUNCTION_NAME and TARGET_ENVS
- task: PowerShell@2
  displayName: 'Parse merge commit for Function App and Env'
  inputs:
```

```
  inputs:
    targetType: 'inline'
    script: |
      $ErrorActionPreference = "Stop"

      Write-Host "Build.SourceVersionMessage:"
      Write-Host "$(Build.SourceVersionMessage)"

      # Try to read last commit message if Build.SourceVersionMessage exists
      $commitMessage = "$(Build.SourceVersionMessage)"
      if ([string]::IsNullOrEmpty($commitMessage) -or $commitMessage -eq 'None') {
        # fallback: use Git to get last commit message
        $commitMessage = git log -1 --pretty=%B
      }
      Write-Host "Commit message:"
      Write-Host $commitMessage

      # Regex: Function App - "name"
      if ($commitMessage -match 'Function App\s*-\s*"?([^"\n\r]+)"?') {
        $functionName = $matches[1].Trim()
        Write-Host "##vso[task.setvariable variable=FUNCTION_NAME]$functionName"
        Write-Host "FUNCTION_NAME: $functionName"
      } else {
        Write-Error "FUNCTION_NAME not found in commit message"
        exit 1
      }

      # Regex: Environment - "Dev" or "Dev,QA" etc
      if ($commitMessage -match 'Environment\s*-\s*"?([^"\n\r]+)"?') {
        $envs = $matches[1].Trim()
        Write-Host "##vso[task.setvariable variable=TARGET_ENVS]$envs"
        Write-Host "TARGET_ENVS: $envs"
      } else {
        Write-Error "TARGET_ENVS not found in commit message"
        exit 1
      }

# 2) Build step (placeholder) - insert your actual build steps
- script: |
    echo "Build placeholder: build code, run tests, produce artifacts"
    mkdir -p $(Build.ArtifactStagingDirectory)/app
    echo "artifact content" > $(Build.ArtifactStagingDirectory)/app/content.txt
  displayName: 'Build/Package - placeholder'

# 3) Publish artifact
- task: PublishPipelineArtifact@1
  inputs:
    targetPath: '$(Build.ArtifactStagingDirectory)'
    artifact: 'drop'

# 4) Wait for any in-progress build on branch/main to finish (optional)
#    Here we ensure that if some other builds are in progress for main, we wait a short while.
#    Adjust polling strategy if needed.
- task: PowerShell@2
  displayName: 'Wait if other builds for main are running (simple)'
  inputs:
    targetType: 'inline'
    script: |
      $project = "$(System.TeamProject)"
      $orgUrl = "$(System.CollectionUri)"
      $token = "$(System.AccessToken)"
      $headers = @{ Authorization = "Bearer $token" }

      $definitionsUrl = "$orgUrl$project/_apis/build/builds?definitions=&reasonFilter=&statusFilter=inProgress&branchName=refs/heads
      # We poll briefly — if you want a longer wait logic implement exponential backoff
      $runs = Invoke-RestMethod -Uri $definitionsUrl -Headers $headers -Method Get
```

```
      if ($runs.count -gt 0) {
        Write-Host "Found $($runs.count) running build(s) on main. Waiting 30s..."
        Start-Sleep -Seconds 30
      } else {
        Write-Host "No running builds on main"
      }

# 5) Trigger the release (create release via REST)
- task: PowerShell@2
  displayName: 'Trigger Azure_build_pipelines Release'
  inputs:
    targetType: 'filePath'
    filePath: .azure-pipelines/trigger_release.ps1
    arguments: '-ReleaseDefinitionId $(releaseDefinitionId) -BuildId $(Build.BuildId) -FunctionName "$(FUNCTION_NAME)" -TargetEnvs "
  env:
    SYSTEM_ACCESSTOKEN: $(System.AccessToken)
```

Notes on `ci-main.yml`:

- It reads the merge commit message (merge commits generally include PR title/description).
- If `Build.SourceVersionMessage` isn't present, it falls back to `git log -1`.
- It publishes artifact `drop`.
- It calls `trigger_release.ps1` to create a release.

---

2) Release trigger script — `.azure-pipelines/trigger_release.ps1`

This script creates a release via the Azure DevOps Release REST API and sets release variables that the release will use. It attaches the `buildId` as an artifact (so release uses that build's artifacts).

```
# .azure-pipelines/trigger_release.ps1
param(
  [Parameter(Mandatory=$true)][string]$ReleaseDefinitionId,
  [Parameter(Mandatory=$true)][string]$BuildId,
  [Parameter(Mandatory=$true)][string]$FunctionName,
  [Parameter(Mandatory=$true)][string]$TargetEnvs
)

$ErrorActionPreference = "Stop"

$orgUrl = "$(System.CollectionUri)" -replace '/$',''   # e.g. https://dev.azure.com/yourorg
$project = "$(System.TeamProject)"
$token = $env:SYSTEM_ACCESSTOKEN
$headers = @{
  Authorization = "Bearer $token"
  'Content-Type' = 'application/json'
}

# Build the JSON body for creating a release
$body = @{
  definitionId = [int]$ReleaseDefinitionId
  description  = "Automated release for Function:$FunctionName Envs:$TargetEnvs (build $BuildId)"
  isDraft      = $false
  artifacts    = @(
    @{
      alias = "_Azure_Master_Build"
      instanceReference = @{
        id = $BuildId
        name = $BuildId
      }
    }
  )
  variables = @{
    FUNCTION_NAME = @{ value = $FunctionName }
    TARGET_ENVS   = @{ value = $TargetEnvs }
    TRIGGERED_BY  = @{ value = "ci-main" }
    REQUEST_TIME  = @{ value = (Get-Date).ToString("o") }
  }
} | ConvertTo-Json -Depth 8

$uri = "$orgUrl/$project/_apis/release/releases?api-version=6.0"

Write-Host "Creating release (definition $ReleaseDefinitionId) for function $FunctionName ..."
$response = Invoke-RestMethod -Method Post -Uri $uri -Headers $headers -Body $body
Write-Host "Release created: id=$($response.id), name=$($response.name)"
Write-Host "Release url: $orgUrl/$project/_release?releaseId=$($response.id)"
```

Notes:

- This script uses `$(System.AccessToken)` for auth. Make sure the calling pipeline job enabled OAuth access.
- `artifacts.alias` must match the artifact alias configured in the release definition — update `_Azure_Master_Build` to your actual alias if needed.

---

3) Release pipeline logic — `release/azure_build_release.yml` (conceptual YAML)

You may already have a classic release `Azure_build_pipelines` with 40 agent jobs. The key is: **each job should have a condition that only runs when `FUNCTION_NAME` matches that job's Function App** and `TARGET_ENVS` contains the environment.

Below is a **job template**. You can replicate it for each Function App job and change `appName` and `appEnv` accordingly.

> If you use the classic Release UI, set job-level **Custom condition** to the expression shown.

```
# release/azure_build_release.yml  (conceptual — you can map into classic)
parameters:
  - name: appName
    type: string
```

```yaml
  - name: appEnv
    type: string   # e.g., 'Dev' or 'QA'
  - name: artifactAlias
    type: string
    default: '_Azure_Master_Build'

jobs:
- job: Deploy_${{ parameters.appName }}
  displayName: 'Deploy ${{ parameters.appName }} to ${{ parameters.appEnv }}'
  pool:
    name: 'YourAgentPool'     # choose the pool where your agent sits
  condition: and(succeeded(), eq(variables['FUNCTION_NAME'], '${{ parameters.appName }}'), contains(variables['TARGET_ENVS'], '${{ p
  steps:
    - task: AzureCLI@2
      displayName: 'Start target Function App'
      inputs:
        azureSubscription: '<YOUR_AZURE_SERVICE_CONNECTION>'
        scriptType: 'ps'
        scriptLocation: 'inlineScript'
        inlineScript: |
          $targetApp = '$(FUNCTION_NAME)'
          Write-Host "Starting function app $targetApp ..."
          # fetch resource group
          $rg = az functionapp show --name $targetApp --query resourceGroup -o tsv
          if ([string]::IsNullOrEmpty($rg)) {
            Write-Error "Function app $targetApp not found"
            exit 1
          }
          az functionapp start --name $targetApp --resource-group $rg

    - download: current
      artifact: $(artifactAlias)

    - task: AzureCLI@2
      displayName: 'Deploy artifact to Function App (zip deploy)'
      inputs:
        azureSubscription: '<YOUR_AZURE_SERVICE_CONNECTION>'
        scriptType: 'bash'
        scriptLocation: 'inlineScript'
        inlineScript: |
          set -e
          TARGET_APP="$(FUNCTION_NAME)"
          RG=$(az functionapp show --name "$TARGET_APP" --query resourceGroup -o tsv)
          if [ -z "$RG" ]; then
            echo "Function app $TARGET_APP not found"
            exit 1
          fi

          # Adjust path if your artifact is zipped differently
          ARTIFACT_PATH="$(Pipeline.Workspace)/drop/app"
          if [ ! -d "$ARTIFACT_PATH" ]; then
            echo "Artifact path not found: $ARTIFACT_PATH"
            exit 1
          fi

          # Zip and deploy
          ZIPFILE="/tmp/${TARGET_APP}.zip"
          cd "$ARTIFACT_PATH"
          zip -r "$ZIPFILE" .
          echo "Deploying $ZIPFILE to $TARGET_APP ..."
          az functionapp deployment source config-zip --name "$TARGET_APP" --resource-group "$RG" --src "$ZIPFILE"

    - task: Bash@3
      displayName: 'Health check'
      inputs:
```

```
      targetType: 'inline'
      script: |
        set -e
        TARGET_APP="$(FUNCTION_NAME)"
        # Assuming function has a health endpoint, adjust as needed
        url="https://${TARGET_APP}.azurewebsites.net/api/health"
        echo "Checking $url ..."
        if ! curl -sSf "$url"; then
          echo "Health check failed!"
          exit 1
        fi

  - task: PowerShell@2
    displayName: 'Notify on success'
    condition: succeeded()
    inputs:
      targetType: 'inline'
      script: |
        Write-Host "Deployment succeeded for $(FUNCTION_NAME) in env $(TARGET_ENVS)"

  - task: PowerShell@2
    displayName: 'Notify on failure'
    condition: failed()
    inputs:
      targetType: 'inline'
      script: |
        # Use your SMTP or other method, or rely on Azure DevOps notifications
        $msg = "Deployment FAILED for $(FUNCTION_NAME) in env $(TARGET_ENVS). See release logs."
        Write-Host $msg
        # Optionally send email via Send-MailMessage using your SMTP server (add SMTP details)
```

How to apply in classic release UI:

- For each agent job, set **"Run this job" → Custom condition** with expression:

  ```
  and(succeeded(), eq(variables['FUNCTION_NAME'], 'bulksaandprocessor-qa-1'), contains(variables['TARGET_ENVS'], 'QA'))
  ```

- Replace `'bulksaandprocessor-qa-1'` and `'QA'` with that job's app name and env.

Concurrency, waiting for builds, and priority handling

Wait for build to finish / ensure build artifacts are ready

- Our `ci-main.yml` waits/polls briefly to ensure other in-progress builds on `main` are not overlapped; more robust approach:
  - The pipeline that triggers release should only trigger the release **after the build finishes successfully**, which is what `ci-main.yml` does — it triggers the release at the end of the successful build job.
  - The created release uses that build's `buildId` for artifact instanceReference, ensuring the release deploys artifacts produced by this build.

Concurrency & Priority for multiple simultaneous PR merges

- Azure DevOps **Environments** provide concurrency control. Configure the target Release **Environment** with:
  - **Deployment queueing**: set **"Concurrency"** to `1` (only one deployment at a time). This causes subsequent releases to *queue*.
  - Deployments to that environment are processed FIFO (Azure DevOps respects release creation order).
- Because the CI pipeline triggers release creation only after successful build and the release creation includes `REQUEST_TIME`, the order in which releases were created determines priority.
- This enforces **time-based priority** without extra external queueing.

How to map to your current classic `Azure_build_pipelines` with ~40 agents

- **Keep your existing release** `Azure_build_pipelines` as-is.
- For each agent/job (Account Data, Add Asset, etc.) update the job condition to the expression that compares `$(FUNCTION_NAME)` and `$(TARGET_ENVS)` as shown above.
- In the job steps, before deployment, also call `az functionapp start` to ensure the Function App is enabled. (We provided code above.)

Security & Permissions checklist (must-do)

1. In the pipeline that uses `$(System.AccessToken)`, enable **Allow scripts to access the OAuth token** in the agent job options.
2. Service connection (`<YOUR_AZURE_SERVICE_CONNECTION>`) must have contributor or necessary-level access to Function Apps in subscription / RG (least privilege: only the RG containing the apps).
3. Restrict who can edit pipelines and release definitions.
4. Use Key Vault for any secrets (SMTP creds etc) — avoid hardcoding.

Notifications

Use one (or both) of:

- Azure DevOps **Project Notifications** (recommended) — create a custom subscription for **Release failed** and target recipients.
- In-pipeline email via `Send-MailMessage` or Office365 connector (requires enabling and storing creds in secure pipeline variables/Key Vault).

Example: How developer should create PR

PR description example:

```
Title: Function-App Deployment: bulksaandprocessor-qa-1

Description:
Implemented changes for bulk sand processor.

Function App - "bulksaandprocessor-qa-1"
Environment - "QA"            # or "Dev,QA" if both
```

When approved and merged, the merge commit includes this description, and `ci-main.yml` extracts values.

Final Notes & next steps for me to help you deploy this

1. I can provide:
    - A script which auto-generates job conditions for all your 40 jobs (so you don't manually edit each job).
    - A version of `ci-main.yml` tailored to your build steps (e.g., if your builds are multi-artifact).
    - A classic release JSON snippet to PATCH your existing release if you want to automate applying these conditions.
2. Confirm these details and I'll generate any of the above:
    - `RELEASE_DEF_ID` (I can guide how to find it) — or I can show you how to discover.
    - Artifact alias in your release (the alias used for the build artifact in your release).
    - Do you want me to produce an exact job-level condition list for all 40 apps? If yes, provide a CSV list of app names and their envs or confirm they follow naming pattern.
3. Want me to produce a zip containing all files and example PR commit message for testing? I can produce the full set ready to check in.

If you want, I'll now:

- Produce the **exact YAML + PowerShell files** populated with placeholders replaced by your real values (if you give them), or
- Produce a script to **auto-patch** all release jobs to set conditions based on job names.

Which next step do you want me to do?