**Immutable Image Generation Architecture**

**🔧 Objective**

Establish a consistent, automated, and secure pipeline to generate immutable VM images using **Azure Image Builder (AIB)**, integrating GitHub, Azure DevOps (ADO), and CDK for Terraform (CDKTF) modules within the **BMO One DevOps** ecosystem.

**📌 Step-by-Step Architecture Flow**

**1. Developer Creates New ADO Project and GitHub Requests in BMO One DevOps Portal**

* **Platform**: BMO One DevOps Portal
* **Artifacts Created**:
  + Azure DevOps (ADO) Project
  + 2 GitHub Repositories
* **Purpose**:
  + One repo for **Cloud Infrastructure provisioning**
  + One repo for **Immutable Image definition and pipelines**

🔒 *Note: Repository requests follow internal security policies and naming conventions enforced by BMO governance teams.*

**2. Cloud Infrastructure Creation using CDKTF (Repo 1)**

* **Repository**: Cloud Infra Repo
* **Tooling**: CDK for Terraform (CDKTF)
* **Modules Used**:
  + bmo\_compute\_gallery\_module
  + bmo\_keyvault\_module
  + bmo\_storage\_account\_module
  + bmo\_resource\_group\_module
* **Infra Components Created**:
  + **Azure Compute Gallery**: Stores custom image versions
  + **Key Vault**: Stores secrets such as AIB service principal secrets
  + **Storage Account**: Stores AIB customization and validation scripts, logs, and staging artifacts
  + **Resource Group**: Logical grouping of resources for AIB builds
* **Branching Strategy**: Standardized main/dev branches with pull request validations

✅ *Output: Cloud infra environment ready to support image build operations*

**3. Access Request for SPN and Managed Identity**

* **Actors**: Developer and Access Management Team
* **Request Type**: Role-Based Access Control (RBAC)
* **Resources Accessed**:
  + Key Vault (Reader / Secret User)
  + Compute Gallery (Contributor)
  + Storage Account (Contributor)
  + Resource Group (Reader/Contributor based on AIB permissions)
* **Identities Created**:
  + **Immutable SPN** (Service Principal Name) – used by AIB for build operations
  + **Managed Identity** – assigned to the pipeline or builder VM if using User-Assigned Managed Identity (UAMI)

🔐 *Ensure minimum required permissions per principle of least privilege.*

**4. CIO Immutable Image Repository Setup (Repo 2)**

* **Purpose**: Defines the custom image and build process
* **Initial Bootstrap**:
  + baseline-config folder: Contains common AIB templates and scripts
  + customization-scripts/: Developer-added PowerShell, Bash, or DSC scripts for image customization
  + validation-scripts/: Optional validation checks post customization
  + ado-pipelines/: YAML-based pipelines to trigger the image creation
* **Folder Structure Example**:
* /
* ├── ado-pipelines/
* │ └── image-builder-pipeline.yml
* ├── baseline-config/
* │ └── default-aib-template.json
* ├── customization-scripts/
* │ └── install-custom-software.ps1
* ├── validation-scripts/
* │ └── validate-win-update.ps1
* └── README.md
* **Pipeline Configuration**:
  + Parameterized to support multiple environments (e.g., dev/test/prod)
  + Integrates with AIB via REST API or az CLI
  + Stores image output to **Azure Compute Gallery** under specific versioning scheme

📦 *Each image version is automatically tagged with version numbers and build metadata.*

**5. Link ADO with GitHub Repo (for Image Build)**

* **Action**: Developer configures a GitHub service connection in Azure DevOps
* **Service Connection**:
  + OAuth or Personal Access Token (PAT)
  + Grants ADO access to CIO Image Repo
* **Security Note**: Use organization-scoped connections with fine-grained permissions
* **Repo Access Includes**:
  + Triggering pipelines on commits
  + Access to YAML pipeline definitions
  + Reading and storing logs

🔁 *CI/CD pipeline becomes active and auto-triggered based on repo commits.*

**6. Pipeline Execution for Image Build**

* **Trigger**: Manual or GitHub Push Trigger
* **Pipeline Tasks**:
  1. Clone CIO image repo
  2. Validate baseline and customization scripts
  3. Upload scripts to Azure Storage
  4. Generate AIB Template (JSON) dynamically or from static template
  5. Submit AIB Build
  6. Monitor image creation process
  7. Publish resulting image to Azure Compute Gallery
* **Key Pipeline Outputs**:
  1. AIB Run logs
  2. Custom image URI (Gallery image version ID)
  3. Notification to stakeholders (via email/Teams/webhook)

📊 *Pipeline status and logs are stored in ADO and optionally in centralized log analytics.*

**Diagram Block Representation (based on your flow)**

[Developer]

│

├──> [BMO One DevOps Portal]

│ └──> Create ADO Project + GitHub Repos

│

├──> [GitHub Repo 1: Cloud Infra]

│ └──> CDKTF Modules → Azure Resources

│ ├── Resource Group

│ ├── Compute Gallery

│ ├── Key Vault

│ └── Storage Account

│

├──> [Access Request for SPN + Managed Identity]

│ └──> RBAC roles applied to cloud resources

│

└──> [GitHub Repo 2: CIO Immutable Image]

├── ADO Pipelines Configured

├── Scripts (Customization + Validation)

└── Baseline AIB Template

[ADO Pipeline]

└──> Uses GitHub Repo 2 → Triggers AIB → Builds Image → Publishes to Azure Compute Gallery

**🔄 Continuation from Step 6: Developer Runs the Pipeline**

**6. Developer Runs the ADO Pipeline**

* **Trigger Method**: Manually from the ADO UI
* **User Interaction**:
  + A **boolean parameter toggle** is provided:  
    ✅ **Patch Base Image?**
    - When enabled: Latest OS updates will be applied using Windows/Linux Update
    - When disabled: Custom image is built from the base without updates
* **Other Pipeline Parameters** (if any):
  + Image Version
  + Customization script location
  + Validation script toggle
  + Environment type (Dev/Test/Prod)

🧩 *Developer has full control over whether the base image is patched or remains as-is.*

**7. Secrets Replacement in Image Build Definition**

* **Build Definition Template**: AIB JSON file, possibly templated using Jinja2 or YAML
* **Action Performed by ADO Script**:
  + Locates placeholders in the template like {{KeyVaultSecret:MySecret}}
  + Uses az keyvault secret show or REST API to fetch values
  + Injects **runtime secret values** into the build template before submission

🔐 *Ensures sensitive information (e.g., admin credentials, tokens) are securely injected only at runtime.*

**8. AIB Image Build Submitted**

* **Submission**: ADO pipeline uses Azure CLI or REST API to submit the final build template to AIB.
* **AIB Infrastructure Provisioned**:
  + 🧠 **Orchestrator VM**: Coordinates the overall build lifecycle
  + 🔧 **Build VM**: Executes customization scripts (software install, hardening, config changes)
  + 🧪 **Validation VM**: Runs post-build validation scripts (e.g., WinRM checks, app test, baseline checks)
* **Parallel Logs Captured**:
  + Time-stamped logs from customization and validation phases
  + Screenshot capture if image build fails on UI-facing scripts (for Windows builds)

🛠️ *The architecture supports modular customization and validation with separate dedicated stages.*

**9. Logs Sent to Storage Account**

* **Log Types Captured**:
  + customization.log
  + validation.log
  + orchestrator.log (optional)
  + AIB telemetry and diagnostic logs
* **Location**: Logs uploaded to a dedicated container in the Storage Account (e.g., aib-logs/<build-id>/)
* **Access Control**:
  + Developer or AIB team has Reader access
  + Logs retained based on lifecycle policies (e.g., 30 or 90 days)

📁 *Both success and failure logs are stored for transparency and traceability.*

**10. Debug Stage (on Failure)**

* **Condition**: If AIB returns a failed build status

**🧪 Debug Pipeline Activated**

* Developer triggers a **“Debug Image Build” pipeline** from ADO
* Actions Performed:
  + Password for Build and/or Validation VM is reset via ADO task (az vm user reset-password)
  + Login information securely shown or emailed
  + Time-bound access window (e.g., 4 hours) for troubleshooting

**🔍 Developer Troubleshoots:**

* Developer accesses the VM via RDP/SSH
* Analyzes logs in real-time or re-runs scripts interactively

**✅ Debug Completion:**

* Once root cause is found:
  + Developer marks Debug stage as **Complete**
  + ADO triggers **cleanup**: Deletes all VMs, disks, NICs, NSGs
  + Fix is committed to GitHub (customization script, parameter, template, etc.)

🧹 *Temporary debug environment is completely destroyed after the session to avoid security risks.*

**11. Successful Build – Image Publishing**

* **Outcome**: If AIB build succeeds
* **ADO Task Actions**:
  1. Captures the successfully built image
  2. Publishes the image version to **Azure Compute Gallery**
     + Tags: imageVersion, buildId, validated, patched
  3. Applies **CMK (Customer-Managed Key) Encryption** to the image
     + Key fetched from Azure Key Vault
     + Attached during az sig image-version create or via AIB template
* **Post-Build Notifications**:
  1. Image version ID shared via Teams/Email/Webhook
  2. GitHub commit status optionally updated (green check)

🛡️ *Resulting image is hardened, encrypted, validated, and ready for downstream consumption.*

**Summary Architecture Flow (Updated)**

[Developer]

└──> Runs ADO Pipeline with Patch Toggle

└──> Secrets injected from Key Vault into AIB Template

└──> Template submitted to AIB

├── Orchestrator VM

├── Build VM → runs customization

└── Validation VM → runs validations

└──> Logs sent to Storage Account

└──> IF FAILED:

├── Trigger Debug Pipeline

├── Reset VM password

└── Troubleshoot and Cleanup

└──> IF SUCCESS:

├── Publish Image to Compute Gallery

└── Apply CMK Encryption