**Azure Immutable Infrastructure - Custom Image Strategy**

**1. Options for Building Custom Images at Scale**

**Options:**

* **Azure Image Builder (AIB) with Azure DevOps (ADO)**
* **Packer with ADO or GitHub Actions**
* **VM Scale Set Custom Images**
* **Containerized Approach (Using ACR & AKS)**

**Analysis:**

| **Option** | **Pros** | **Cons** |
| --- | --- | --- |
| **AIB with ADO** | Seamless Azure integration, automatic updates, cost-effective, and GitHub integration | Requires upfront setup and governance policies |
| **Packer with ADO/GitHub Actions** | Multi-cloud support, better version control, and parallel builds | Requires additional setup for security and automation |
| **VM Scale Set Custom Images** | Good for dynamic scaling workloads | Higher complexity and cost for non-VMSS-based workloads |
| **Containerized Approach** | Easy deployments and scalability | Not suitable for non-containerized applications |

**Recommendation:**

* Use **Azure Image Builder (AIB) with Azure DevOps (ADO)** as it provides a standardized and scalable approach with built-in governance and security controls.

**2. Each CIO Having Their Own GitHub Repo vs. One GitHub Repo for All CIO Custom Images**

**Comparison:**

| **Approach** | **Pros** | **Cons** |
| --- | --- | --- |
| **Each CIO has their own repo** | Autonomy, flexibility, and separate access control | Harder to enforce governance, potential duplication of effort |
| **Single repo for all CIO custom images** | Easier governance, centralized management, and code reuse | Limited flexibility, potential bottlenecks in approvals |

**Recommendation:**

* Use **a single centralized repository** with well-defined branches for each CIO team to ensure consistency and governance while allowing controlled customization.

**3. Each CIO Having Their Own ADO Project/Pipelines vs. One ADO Project for All CIO Custom Images**

**Comparison:**

| **Approach** | **Pros** | **Cons** |
| --- | --- | --- |
| **Each CIO has their own ADO Project** | Independent workflows, better isolation | More administrative overhead, higher maintenance costs |
| **One ADO Project for all CIO custom images** | Centralized governance, shared resources, reduced operational cost | Potential pipeline congestion, need strict role-based access control |

**Recommendation:**

* **Use a single ADO project with separate pipelines per CIO team**, leveraging templates and modularized pipeline definitions for efficiency and security.

**4. Sharing Other Services vs. CIOs Having Their Own Services**

**Comparison:**

| **Service** | **Shared Model** | **Dedicated Model** |
| --- | --- | --- |
| **Key Vault** | Easier access management, cost savings | Better security isolation |
| **Storage Accounts** | Cost-efficient and simplified management | CIOs can have specific security policies |
| **Staging Resource Groups** | Streamlined governance, easier deployment | More control for CIO teams |
| **Azure Service Principals** | Easier rotation, centralized auditing | CIOs have full autonomy |
| **Managed Identities** | Lower administrative overhead | CIOs have specific role assignments |
| **Compute Gallery** | Shared image versions across teams | CIOs can manage their own lifecycle |

**Recommendation:**

* Use **a shared model for cost-efficient and manageable resources** but allow **critical security components (Key Vault, Service Principals) to be CIO-owned** for better governance.

**5. Monthly Base Image Updates & Impact on Custom Images**

**Challenges:**

* Ensuring CIO teams are notified about base image updates.
* Automating rebuilds of custom images when the base image changes.
* Avoiding breaking changes due to application dependencies.

**Solution:**

1. **Automated Base Image Versioning**: Publish new images to the **Compute Gallery** with versioning.
2. **CI/CD Integration**: Trigger **pipeline executions** when a new base image is available.
3. **Dependency Scanning**: Ensure applications are compatible with the new base image before deployment.
4. **Notification System**: Implement ADO notifications to alert CIO teams of updates.
5. **Rollback Strategy**: Maintain previous versions for rollback in case of failures.

**Recommendation:**

* Implement an **automated image update strategy** with **notifications and rollback plans** to ensure smooth transitions.

**Shared Responsibility Model vs. Isolated vs. Centralized Solutions**

**Comparison:**

| **Model** | **Engineering Responsibility** | **CIO Responsibility** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- |
| **Shared Responsibility** | Engineering provides baseline images, tooling, and governance | CIOs build and maintain their images | Best balance of governance and flexibility | Requires clear process definitions |
| **Isolated Model** | Engineering provides only platform guidelines | CIOs own the entire pipeline and services | Full autonomy | Higher costs, duplicate effort |
| **Centralized Model** | Engineering builds and maintains all images | CIOs consume ready-to-use images | Efficient, cost-effective | CIOs have limited flexibility |

**Recommendation:**

* Use the **Shared Responsibility Model**, allowing Engineering to handle **infrastructure, governance, and automation**, while CIO teams focus on **application-specific image customization**.

Here’s the updated document with the recommended changes and additional pros/cons for the responsibility model. I’ve also included a RACI chart covering Azure services involved in the solution.

**Immutable Infrastructure – CIO Custom Image Strategy**

**Background**

We have an Azure Image Builder (AIB) solution implemented for CGIP Images. CGIP images are custom-built images incorporating BMO-specific configurations and tools. The AIB solution integrates with **Azure DevOps (ADO)** for building images and **GitHub** for storing code related to customizations, templates, and compute gallery metadata.

This project aims to enable CIOs to create **Immutable Infrastructure** by leveraging CGIP Images as a baseline and customizing them with application-specific runtimes, binaries, and dependencies. The resulting images will be built using AIB and managed within the Azure ecosystem.

**1. Options for Building a Custom Image at Scale**

**Approach 1: Using Azure Image Builder (AIB)**

* **Method**: AIB builds a new image by taking a **base CGIP image** and layering on additional software from sources like Artifactory.
* **Scaling Mechanism**: AIB pipelines in Azure DevOps can be configured to execute on demand or scheduled basis.

✅ **Pros**:

* **Automated & Repeatable**: Streamlines image creation via DevOps pipelines.
* **Version Control**: Images are published to **Azure Compute Gallery**, ensuring traceability.
* **Security & Compliance**: Uses **managed identities** to interact with storage, Artifactory, and other services securely.

❌ **Cons**:

* **Build Time**: Image builds take time (~30-60 mins depending on customization).
* **Debugging Complexity**: Issues in provisioning scripts or package installations require troubleshooting AIB logs.
* **Resource Utilization**: Temporary VMs used for image builds may lead to cost spikes.

**2. GitHub Repository Structure: Separate Repo for Each CIO**

**Approach 1: Separate Repository Per CIO (Recommended)**

* Each CIO team has its **own dedicated GitHub repository** for storing application-specific scripts, build definitions, and configurations.

✅ **Pros**:

* **Autonomy**: CIO teams can independently manage and maintain their image configurations.
* **Security & Access Control**: Limits exposure to other teams' repositories.
* **Customization**: Teams can implement different workflows per their application requirements.

❌ **Cons**:

* **Management Overhead**: Multiple repositories require governance and maintenance.
* **Duplication**: Similar configurations may exist across different CIO repositories, leading to inefficiencies.

**3. Azure DevOps Structure: Separate ADO Pipelines for Each CIO vs. Centralized Pipelines**

**Approach 1: Separate ADO Pipelines Per CIO**

* Each CIO team has its own **Azure DevOps project** with independent pipelines.

✅ **Pros**:

* **Autonomy & Customization**: CIO teams can tailor pipelines to their needs.
* **Security**: Ensures each team’s builds are isolated.

❌ **Cons**:

* **Duplication of Effort**: Each team must configure and maintain their pipelines.
* **Inefficient Resource Utilization**: Increases ADO service usage and maintenance costs.

**Approach 2: Centralized ADO Pipelines (Recommended)**

* A single ADO project contains pipelines for **all CIO custom images**, utilizing parameters to differentiate builds.

✅ **Pros**:

* **Standardization**: Ensures consistency in image-building processes.
* **Reduced Overhead**: Fewer ADO projects to maintain.
* **Efficient Resource Utilization**: Shared pipeline agents optimize cost.

❌ **Cons**:

* **Access Management Complexity**: Requires robust RBAC policies.
* **Pipeline Customization Limitations**: All teams must follow the same framework.

**4. Shared vs. Dedicated Azure Services**

**Recommendation: Shared Services with Separate Compute Gallery per CIO**

| **Azure Service** | **Model** |
| --- | --- |
| **Key Vault** | Shared |
| **Storage Accounts** | Shared |
| **Staging Resource Groups** | Shared |
| **Azure Service Principals** | Shared |
| **Managed Identity** | Shared |
| **Azure Compute Gallery** | Separate for each CIO |

✅ **Pros**:

* **Optimized Cost & Governance**: Shared Key Vault, storage, and identities reduce duplication.
* **Security & Isolation**: Separate Compute Gallery ensures CIO-specific image management.

❌ **Cons**:

* **Governance Complexity**: Requires clear RBAC policies to avoid security risks.
* **Dependency on Centralized Engineering Team**: CIO teams rely on Engineering for shared resources.

**5. Base Image Updates: Manual CIO-Managed Updates**

* **Approach**: When a **new CGIP base image** is released, CIO teams will be **notified** but retain control over when to rebuild their CIO custom images.

✅ **Pros**:

* **Controlled Update Process**: CIO teams can test changes before adopting the new base image.
* **Reduced Build Failures**: Avoids forced updates that may break application dependencies.

❌ **Cons**:

* **Risk of Outdated Images**: Some CIO teams may delay updates, leading to security risks.
* **Manual Effort**: Requires periodic review by CIO teams.

**Shared vs. Isolated Responsibility Model**

| **Responsibility Model** | **Pros** | **Cons** |
| --- | --- | --- |
| **Shared Model (Recommended)** | Lower cost, streamlined governance, centralized security, reduced duplication | Requires strong governance, dependency on Engineering |
| **Fully Isolated (CIO-Owned Services)** | CIOs have full autonomy, independent timelines | Higher cost, duplication of services, security challenges |
| **Centralized Model (Engineering Manages Everything)** | Simplified management, security compliance, economies of scale | CIO teams have less control, delays in customization |

**RACI Chart: Azure Services and Responsibility Allocation**

| **Azure Service** | **Engineering Team** | **CIO Teams** |
| --- | --- | --- |
| **Azure Compute Gallery** | C (Provides Baseline) | R/A (Manages CIO Images) |
| **Azure Image Builder** | R/A | C |
| **GitHub Repo** | C | R/A |
| **Azure DevOps Pipelines** | R | A |
| **Key Vault** | R | C |
| **Storage Account** | R | C |
| **Managed Identity** | R | C |
| **Service Principals** | R | C |
| **Base CGIP Image Updates** | A | C |
| **Custom Image Build Triggers** | C | R/A |
| **Security Compliance** | R | A |

* **R** = Responsible
* **A** = Accountable
* **C** = Consulted

**Final Recommendations**

1. **Custom Image at Scale**: Use AIB with Azure DevOps for automation.
2. **GitHub Structure**: **Separate repo per CIO** for better isolation.
3. **Azure DevOps Structure**: **Centralized ADO pipelines** for efficiency.
4. **Shared Services Model**: Use shared Key Vault, storage, and identities but **separate Compute Gallery per CIO**.
5. **Base Image Updates**: CIOs get notified of new CGIP base images but control their rebuild schedule.

**Immutable Infrastructure – CIO Custom Image Strategy**

**Background**

We have an Azure Image Builder (AIB) solution implemented for CGIP Images. CGIP images are custom-built images incorporating BMO-specific configurations and tools. The AIB solution integrates with **Azure DevOps (ADO)** for building images and **GitHub** for storing code related to customizations, templates, and compute gallery metadata.

This project aims to enable CIOs to create **Immutable Infrastructure** by leveraging CGIP Images as a baseline and customizing them with application-specific runtimes, binaries, and dependencies. The resulting images will be built using AIB and managed within the Azure ecosystem.

**Key Considerations & Challenges for Design Phase**

**1. AIB Dependency on Multiple Azure Services**

* AIB relies on several Azure services, including:
  + **Key Vault** for secrets management
  + **Storage Accounts** for storing artifacts
  + **Azure Networking** for connectivity
  + **Azure Policies** for compliance enforcement
  + **Azure Compute Galleries** for image storage
  + **Azure DevOps, GitHub, and Artifactory** for automation and package management
* **Challenge**: Downtime in any of these services can cause AIB builds to fail, leading to delays in custom image creation.
* **Mitigation**: Implement retry mechanisms and define fallback strategies where applicable.

**2. Increased Cost for Shared Services**

* Currently, **shared services** are primarily used for CGIP image builds. Expanding usage to CIO teams will **increase storage, network, and compute costs**.
* **Challenge**: Cost optimization needs to be considered, as billing will be impacted due to high utilization.
* **Mitigation**:
  + Monitor shared resource consumption using **Azure Cost Management**.
  + Implement **quotas and rate limits** to prevent excessive usage.
  + CIO teams should **fund additional capacity** if usage exceeds shared service limits.

**3. ADO Pipeline Resource Constraints & Build Cancellation Policy**

* **Azure DevOps Team Recommendation**: Any builds running **over 2 hours** should use a **dedicated pool** rather than the shared pool.
* **Challenge**: Builds exceeding 2 hours in the shared pool **may be cancelled** unexpectedly, impacting CIO teams.
* **Mitigation**:
  + Identify long-running builds and transition them to a **dedicated agent pool**.
  + CIO teams should estimate build durations and align with **ADO team guidelines**.

**4. Cost Attribution for Shared Services**

* **Orchestration team’s Application Cat ID will be charged** for Shared Services, including ADO usage.
* **Challenge**: Increased billing impact on the orchestration team’s budget.
* **Mitigation**: CIO teams may need to **co-fund shared services** based on usage. **Chargeback mechanisms** should be explored.

**5. Orchestration Team’s Resource Constraints**

* The **Orchestration team** will be responsible for:
  + Initial setup and enablement of the new CIO custom image service.
  + Managing shared services (Key Vault, Storage, Networking, Compute Galleries, etc.).
  + **Troubleshooting failures in AIB builds** when required.
* **Challenge**: The Orchestration team has limited bandwidth to support CIO teams. **Support bottlenecks** could delay resolution.
* **Mitigation**:
  + Define **clear troubleshooting documentation** for CIO teams to handle common issues.
  + CIO teams should be **self-sufficient for minor failures** and escalate to Orchestration only for critical issues.
  + Consider a **support model** (e.g., dedicated SME for escalations).

**Next Steps for Design Phase**

* Review **service dependencies** and define risk mitigation strategies.
* Validate **cost allocation models** and ensure transparency in billing.
* Define **build execution policies** to avoid unexpected cancellations.
* Establish **a self-service troubleshooting model** for CIO teams.
* Align with ADO team on **dedicated pool requirements** for long-running builds.