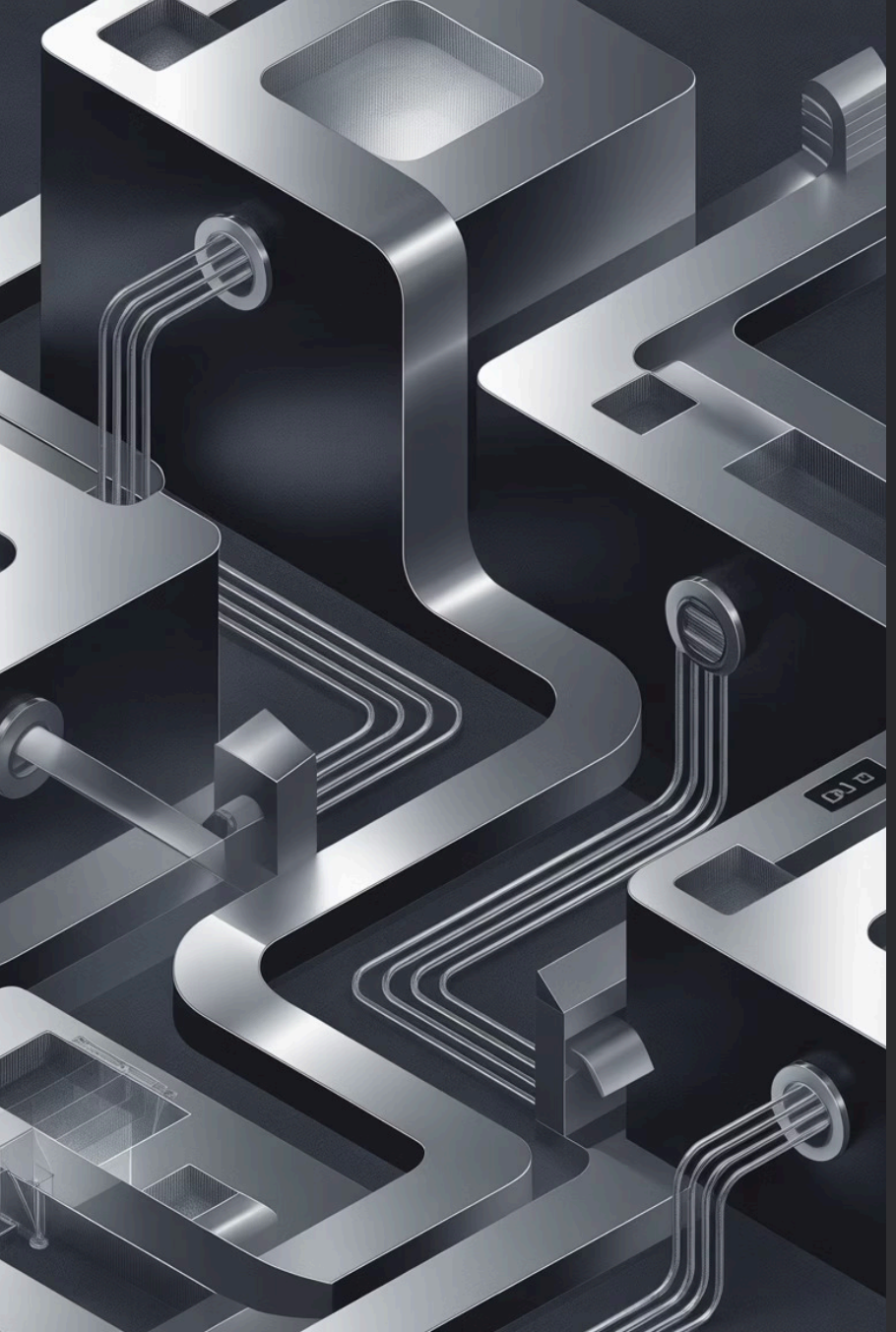


Module 1.3: Backends & API Importing

Mastering the fundamentals of bringing APIs into Azure API Management

Module Overview

Azure API Management provides multiple pathways for importing and configuring APIs, each suited to different scenarios and backend architectures. Understanding these options enables you to establish robust, maintainable API infrastructures that scale with your organization's needs.

01

Backend Types & Architecture

Explore the diverse backend services APIM supports and how they integrate

02

Import Methods & Techniques

Learn multiple approaches to bringing APIs into APIM efficiently

03

Configuration Best Practices

Establish patterns for maintainable, secure backend configurations

Understanding Backend Types in APIM

Azure API Management supports a comprehensive range of backend service types, providing flexibility for diverse architectural patterns. Each backend type offers unique integration capabilities while maintaining consistent management through APIM's unified interface.

HTTP/REST Backends

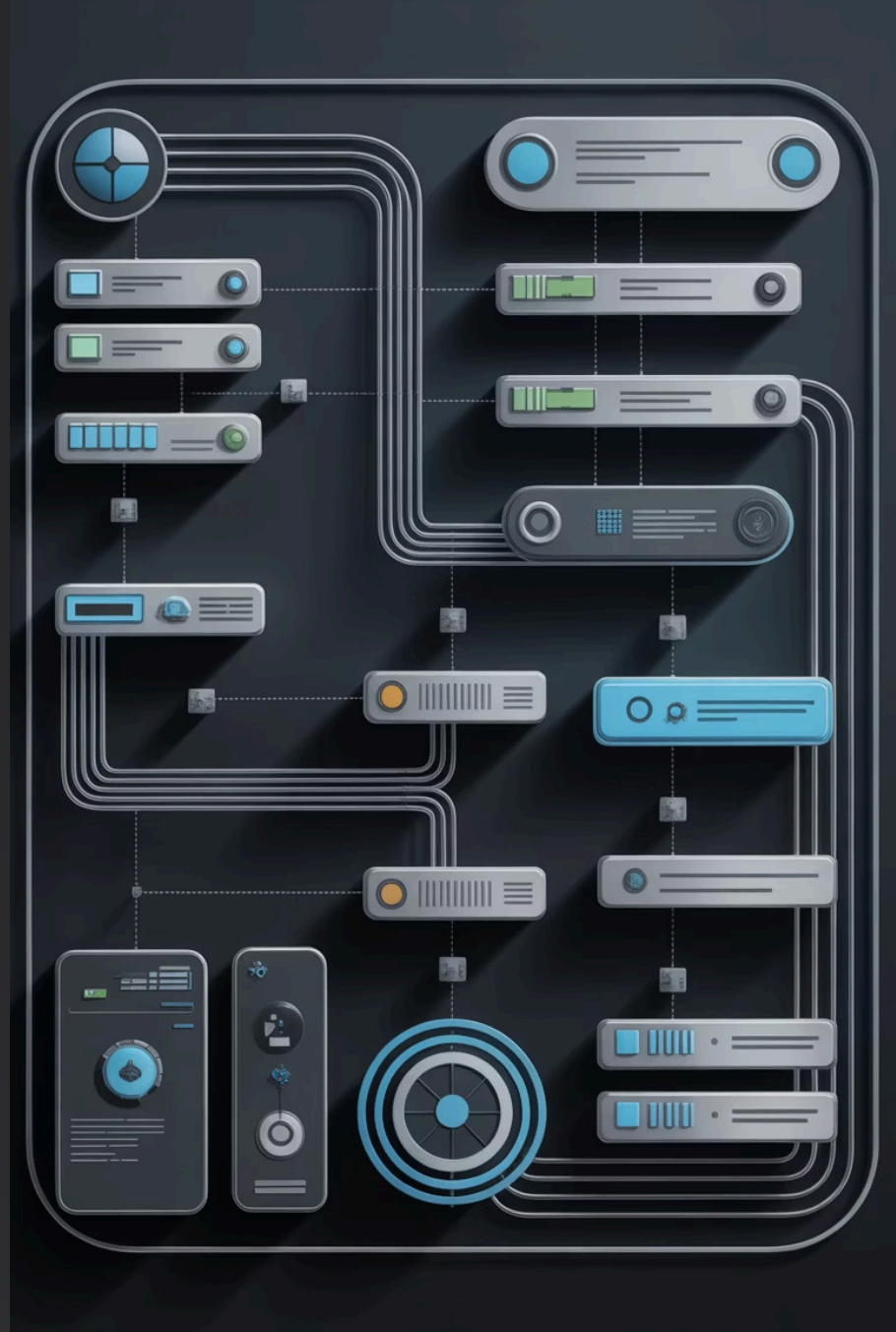
Any HTTP-based service, whether hosted in Azure, on-premises, or third-party providers. The most flexible option for diverse integrations.

Azure Native Services

Function Apps, App Services, Logic Apps, and Container Apps with streamlined authentication and configuration.

Container Services

Azure Kubernetes Service (AKS), Container Instances, and containerized workloads with service mesh integration capabilities.



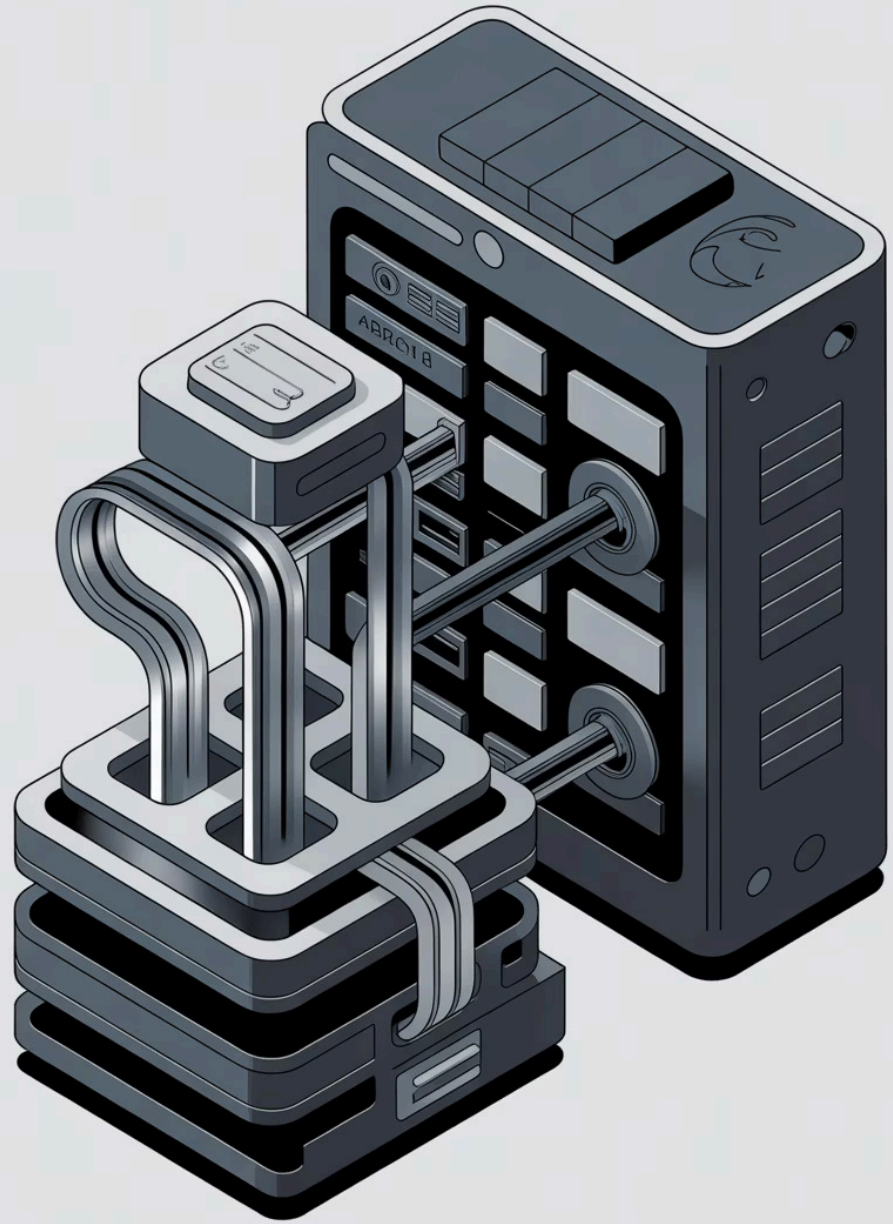
HTTP Backend Architecture

HTTP backends represent the most versatile integration option in APIM, supporting any service that exposes an HTTP or HTTPS endpoint. This approach provides maximum flexibility for hybrid cloud scenarios, legacy system integration, and third-party API consumption.

Key configuration elements include:

- Base URL with protocol, hostname, and optional path prefix
- Connection timeout and retry policies
- Authentication schemes (basic, certificate, OAuth, managed identity)
- Circuit breaker patterns for resilience
- Custom headers and query parameters

Pro Tip: Use named values for base URLs to enable environment-specific configuration without modifying policies.



Azure Function Apps as Backends

Azure Function Apps provide serverless, event-driven backend capabilities with seamless APIM integration. This pairing enables cost-effective API implementations that scale automatically based on demand.

Direct Integration Benefits

- Automatic discovery of function endpoints
- Managed identity authentication without keys
- Function-level authorization control
- Built-in monitoring correlation

Configuration Considerations

- Function host keys vs. system-managed identities
- Cold start mitigation with Premium plans
- OpenAPI extension generation from attributes
- Durable Functions orchestration patterns

App Services & Logic Apps Integration

Azure App Services

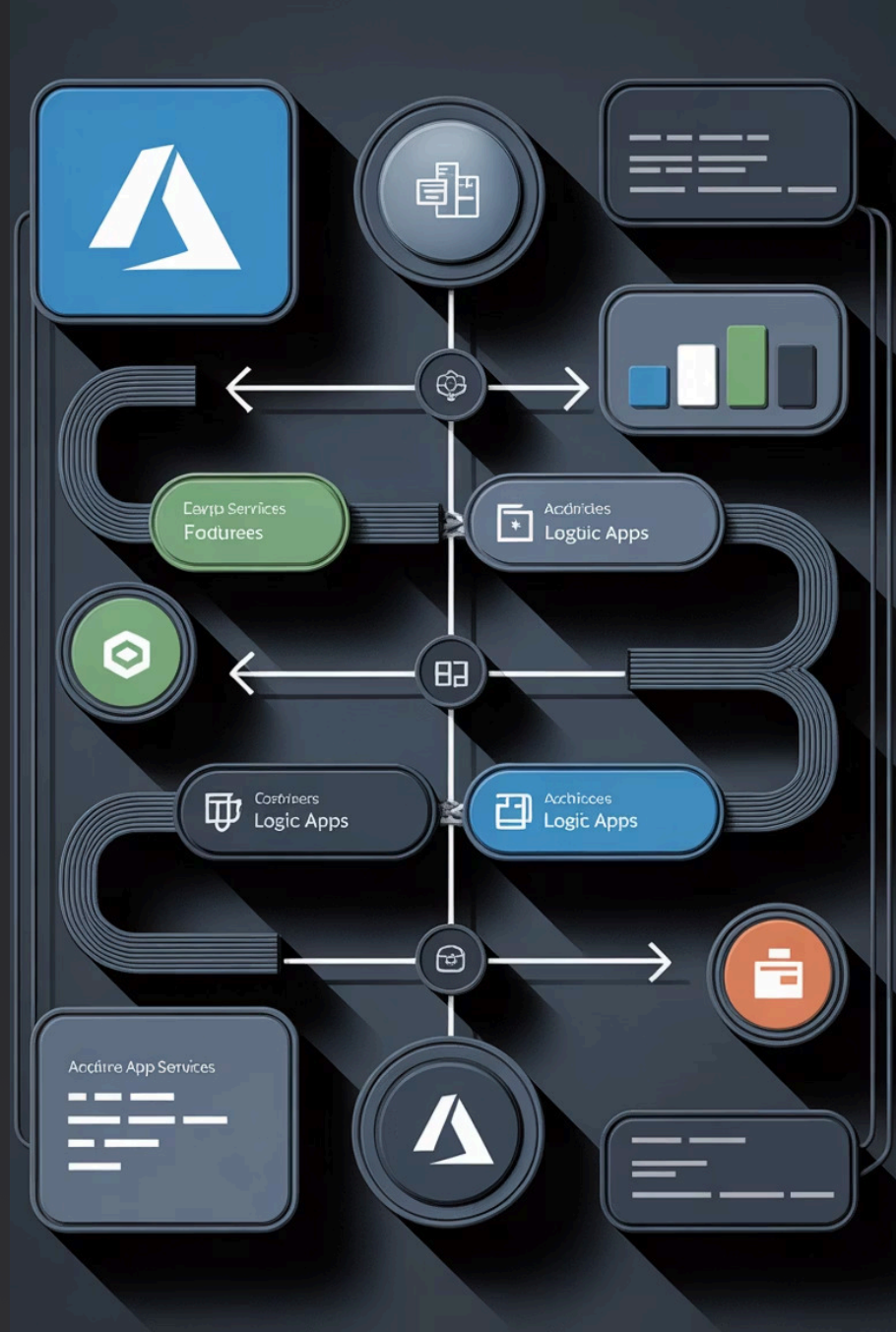
App Services provide a fully managed platform for hosting web APIs with built-in scaling, deployment slots, and comprehensive monitoring. APIM integration enables centralized API governance while leveraging App Service's robust hosting capabilities.

- Automatic SSL/TLS with custom domains
- Deployment slot integration for blue-green deployments
- Authentication provider integration (Easy Auth)
- Private endpoint support for network isolation

Azure Logic Apps

Logic Apps enable low-code workflow orchestration exposed as APIs through APIM. This combination provides powerful integration scenarios with minimal code requirements.

- Visual workflow design with 400+ connectors
- HTTP trigger exposure through APIM
- Enterprise integration pack for B2B scenarios
- Run history and detailed diagnostics



Container & Kubernetes Backends

Azure Kubernetes Service (AKS) and container-based backends provide maximum flexibility for microservices architectures. APIM serves as the API gateway layer, offering consistent management across diverse containerized workloads.

AKS Integration Patterns

Service mesh compatibility with Istio and Linkerd, ingress controller integration, pod-level load balancing, and namespace-based routing strategies.

Container Instances

Lightweight container hosting for serverless scenarios, rapid deployment with minimal infrastructure, integration with virtual networks, and sidecar pattern support.

Service Discovery

Dynamic endpoint resolution using Azure DNS, Consul integration for service mesh, health check automation, and failover configurations.

API Import Methods

Multiple pathways to bring your APIs into APIM

Manual API Creation in Portal

Creating APIs manually through the Azure portal provides complete control over API definitions and is ideal for designing APIs from scratch or when existing specifications are unavailable. This approach allows you to build your API structure iteratively while testing each operation.

Create API Definition

Define API name, URL scheme, path, and versioning strategy in the portal interface

Add Operations

Configure HTTP methods, URL templates, parameters, and request/response schemas

Configure Backend

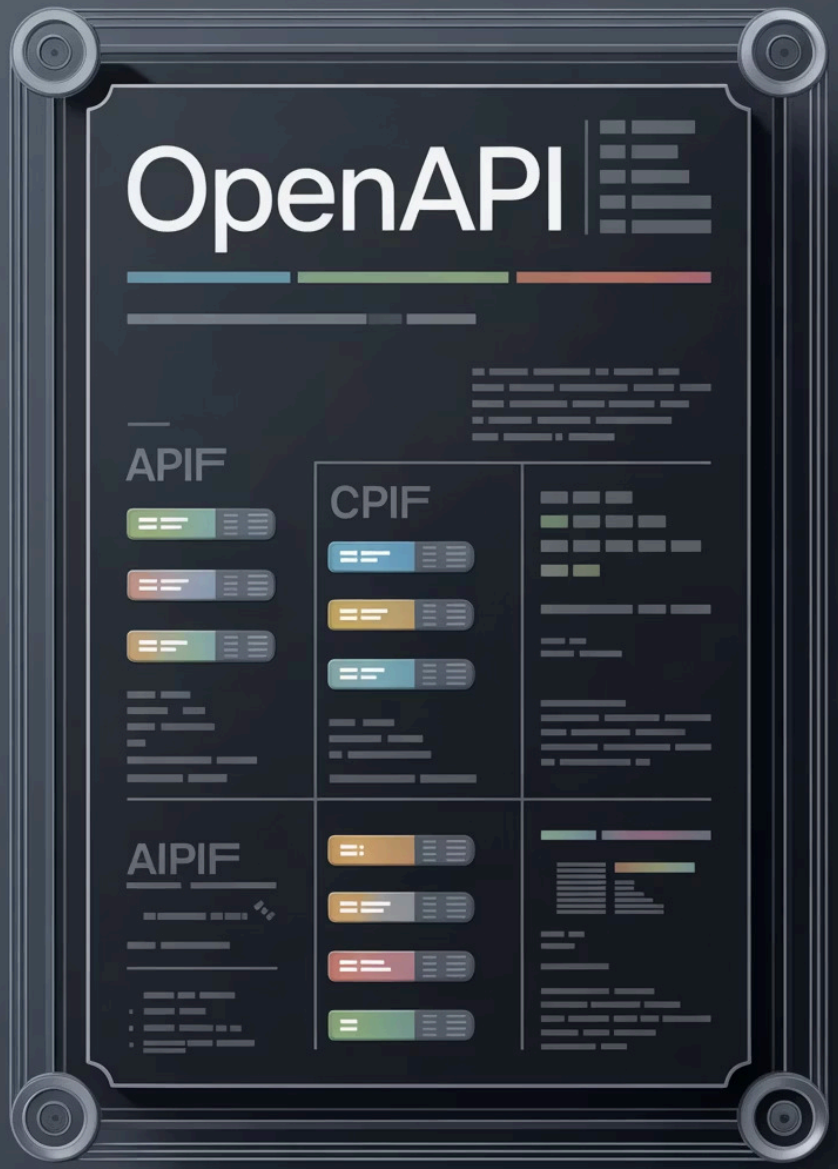
Set backend service URL, authentication, and connection parameters

Apply Policies

Add transformation, security, and routing policies at appropriate scopes

When to Use Manual Creation:

- Designing new APIs
- No existing OpenAPI spec
- Learning APIM concepts
- Prototyping scenarios



Importing from OpenAPI Specifications

OpenAPI (formerly Swagger) import represents the fastest and most accurate method for bringing well-documented APIs into APIM. This approach preserves operation definitions, parameter schemas, response types, and example payloads from your existing API contracts.

OpenAPI 2.0 (Swagger)

Legacy format still widely used. Supports basic RESTful operations, security definitions, and response schemas. Import from JSON or YAML files.

OpenAPI 3.0+

Modern specification with enhanced features including multiple servers, callbacks, improved security schemes, and component reusability. Recommended for new APIs.

Import Sources

Upload local files, reference public URLs, fetch from Git repositories, or extract from Azure resources with built-in OpenAPI generation.

WSDL Import for SOAP Services

While REST APIs dominate modern architectures, many enterprise systems still rely on SOAP-based web services. APIM provides robust WSDL import capabilities, enabling you to expose legacy SOAP services through modern API management patterns.

APIM offers two SOAP integration modes:

- **SOAP pass-through:** Forward SOAP messages directly to backend without transformation, preserving all SOAP envelope details
- **SOAP to REST:** Automatically convert SOAP operations to RESTful endpoints, enabling modern API consumption patterns while maintaining backend compatibility

SOAP-to-REST conversion handles XML-to-JSON transformation, making legacy services accessible to modern clients without backend modifications.

Importing from Azure Resources

Azure native services like Function Apps, App Services, and Logic Apps support direct import into APIM, streamlining the process of exposing these services as managed APIs. This integration leverages Azure's resource metadata and authentication mechanisms for simplified configuration.

Function App Import

Automatically discovers HTTP-triggered functions, imports route templates, and configures authentication with function keys or managed identities.

App Service Import

Connects to running App Services, discovers endpoints from OpenAPI if available, and configures network connectivity through VNet integration or private endpoints.

Logic App Import

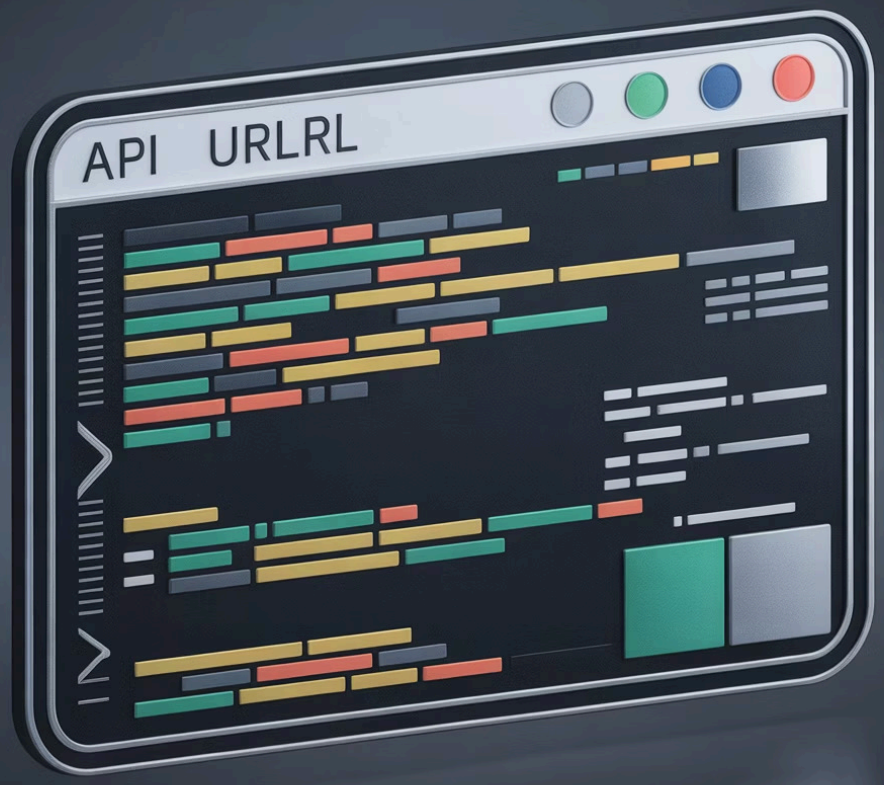
Exposes HTTP-triggered workflows as API operations, maintains connection to Logic App for monitoring, and preserves workflow parameters as API parameters.

Import from URL vs. Git Repository

URL-Based Import

Importing from a public or authenticated URL provides immediate access to OpenAPI specifications hosted anywhere on the internet or your private networks.

- Real-time import from live documentation endpoints
- Supports basic authentication headers
- Ideal for third-party API specifications
- No version control integration
- Manual re-import required for updates



Git Repository Import

Git-based import enables version-controlled API definitions, supporting CI/CD workflows and collaborative API design processes.

- Track changes to API definitions over time
- Branch-based development workflows
- Integration with Azure DevOps or GitHub
- Automated sync with repository updates
- Support for mono-repo and multi-repo strategies

Backend Configuration

Essential settings for robust API backends

Base URL Configuration & Best Practices

The backend base URL serves as the foundation for API routing in APIM. Proper configuration ensures reliable request forwarding while enabling flexibility for environment-specific deployments and infrastructure changes.

URL Structure

Include protocol (http/https), hostname or IP, port (if non-standard), and optional base path. Avoid operation-specific paths in base URL.

Environment Strategy

Use named values or Key Vault references for environment-specific URLs (dev, staging, production) to avoid policy modifications.

Stability Principle

Keep base URLs stable across API versions and revisions. Changes should be rare and carefully managed through deployment pipelines.

Example: Use `https://{{backend-url}}/api` where `backend-url` is a named value, rather than hardcoding https://prod-api.contoso.com/api

Credentials & Authentication Configuration

APIM supports multiple authentication mechanisms for backend services, enabling secure communication while maintaining flexibility for different security requirements. Choosing the appropriate method depends on your backend capabilities and organizational security policies.

1

Basic Authentication

Username and password credentials encoded in request headers. Simple but less secure; use only over HTTPS. Store credentials in Key Vault, not policy XML.

2

Client Certificates

Mutual TLS authentication using X.509 certificates. Provides strong authentication with certificate validation. Upload certificates to APIM or reference Key Vault.

3

Managed Identities

Azure AD-based authentication without credential management. System or user-assigned identities eliminate secrets. Ideal for Azure-native backends.

4

OAuth 2.0 / OpenID Connect

Token-based authentication with authorization servers. Supports delegated access patterns and fine-grained permissions. Configure token acquisition policies.

Client Certificate Deep Dive

Client certificate authentication provides mutual TLS (mTLS), where both client and server authenticate each other using X.509 certificates. This mechanism offers strong security for API-to-API communication and is commonly required in financial services and healthcare scenarios.

Configuration steps in APIM:

1. Upload certificate to APIM (PFX/P12 format with private key)
2. Configure certificate validation settings (issuer, subject, thumbprint)
3. Apply authentication-certificate policy to send certificate to backend
4. Handle certificate renewal and rotation procedures
5. Monitor certificate expiration with Azure alerts

Store certificate passwords in Azure Key Vault and reference them using named values for enhanced security.

Named Values & Key Vault Integration

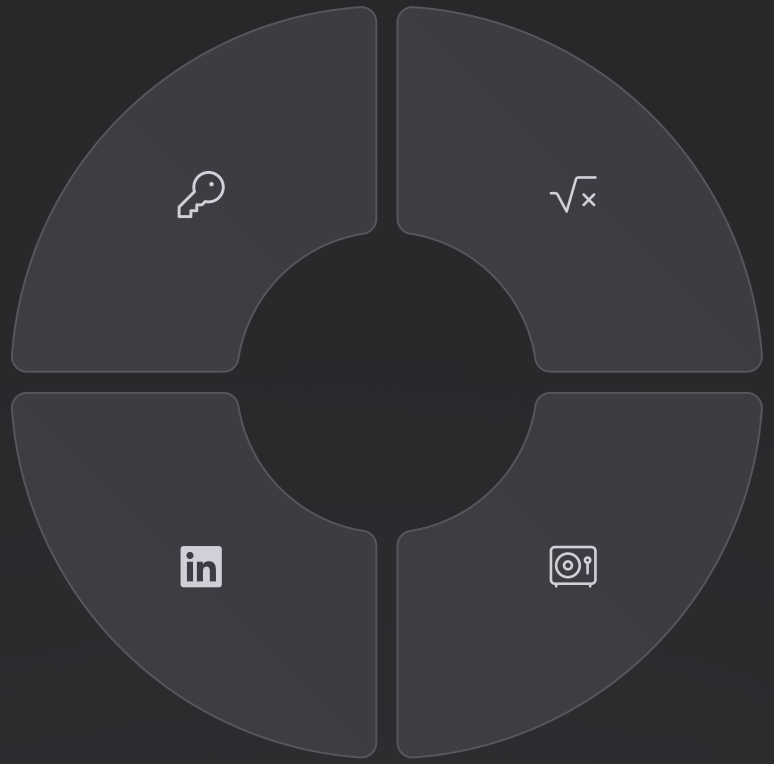
Named values provide a centralized mechanism for managing configuration data, secrets, and environment-specific settings across your APIM instance. Integration with Azure Key Vault adds enterprise-grade secret management with automatic rotation and audit capabilities.

Secret Values

Store sensitive data like API keys, passwords, and connection strings. Encrypted at rest and masked in portal display.

Plain Values

Non-sensitive configuration like URLs, thresholds, and feature flags. Visible and easily editable in portal.

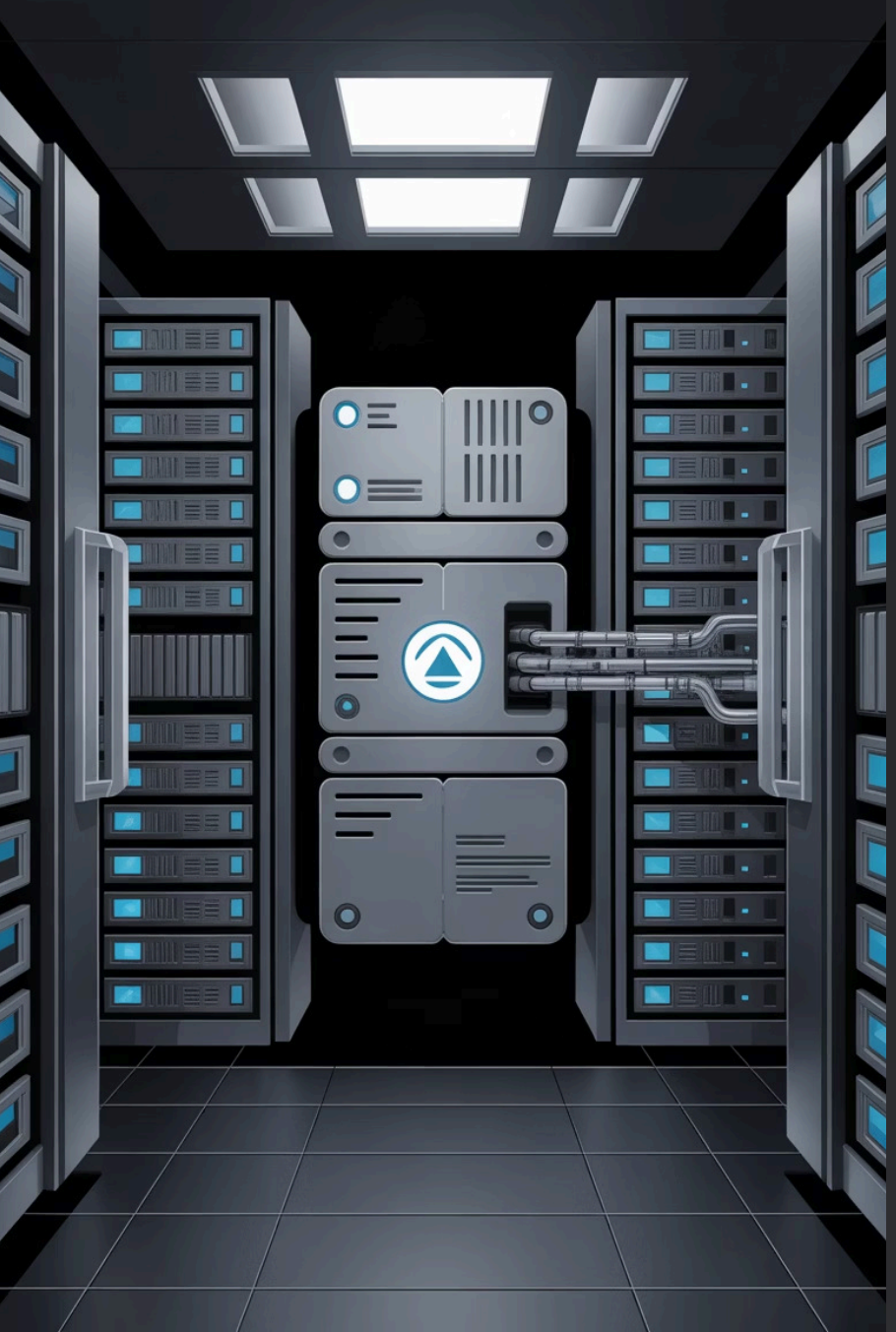


Policy Integration

Reference named values in policies using double-brace syntax: `{{named-value}}`. Simplifies policy maintenance.

Key Vault References

Reference secrets stored in Azure Key Vault. Automatic refresh on updates, centralized rotation, and audit logging.



Key Vault Integration Architecture

Azure Key Vault integration enables APIM to securely access secrets without storing them directly in the service. This architecture pattern provides several critical security benefits while maintaining operational simplicity.

Configuration Requirements

- Enable system-assigned managed identity on APIM instance
- Grant "Get" permission on Key Vault secrets to APIM identity
- Create named value with Key Vault secret identifier URI
- Configure refresh interval for secret rotation

Operational Benefits

- Centralized secret management across multiple services
- Automated secret rotation without APIM downtime
- Comprehensive audit logs for secret access
- Separation of duties between security and API teams

Advanced Backend Configuration

Beyond basic URL and authentication settings, APIM offers sophisticated backend configuration options that enhance reliability, performance, and observability of your API infrastructure.

1

Connection Timeouts

Configure connect and send timeouts separately. Default is 60 seconds; adjust based on backend SLAs. Use shorter timeouts for fast-fail scenarios.

2

Circuit Breaker Patterns

Implement circuit breaker policies to prevent cascading failures. Define failure thresholds, open duration, and recovery attempts.

3

Custom Headers

Add persistent headers to all backend requests (correlation IDs, API versions, routing hints). Use for backend observability and routing logic.

4

Backend Pools

Define multiple backend endpoints with load balancing strategies. Supports failover, round-robin, and weighted distribution patterns.

Import Best Practices

Strategies for maintainable, scalable API management

Stable Base URLs & Configuration Management

Maintaining stable backend base URLs is foundational to operational excellence in APIM. Changes to base URLs require policy updates across multiple APIs and can introduce deployment risks. Following these practices minimizes disruption and maximizes flexibility.

Use Named Values

Abstract base URLs into named values referenced in policies. Update once, affect all APIs. Supports environment promotion without policy changes.

DNS Abstraction

Use DNS names rather than IP addresses. Enables infrastructure changes without APIM reconfiguration. Leverage Azure Private DNS for internal services.

Version Independence

Keep base URL independent of API version numbers. Version information should be in API path, not backend URL. Simplifies multi-version support.

Versioning & Tags from Day One

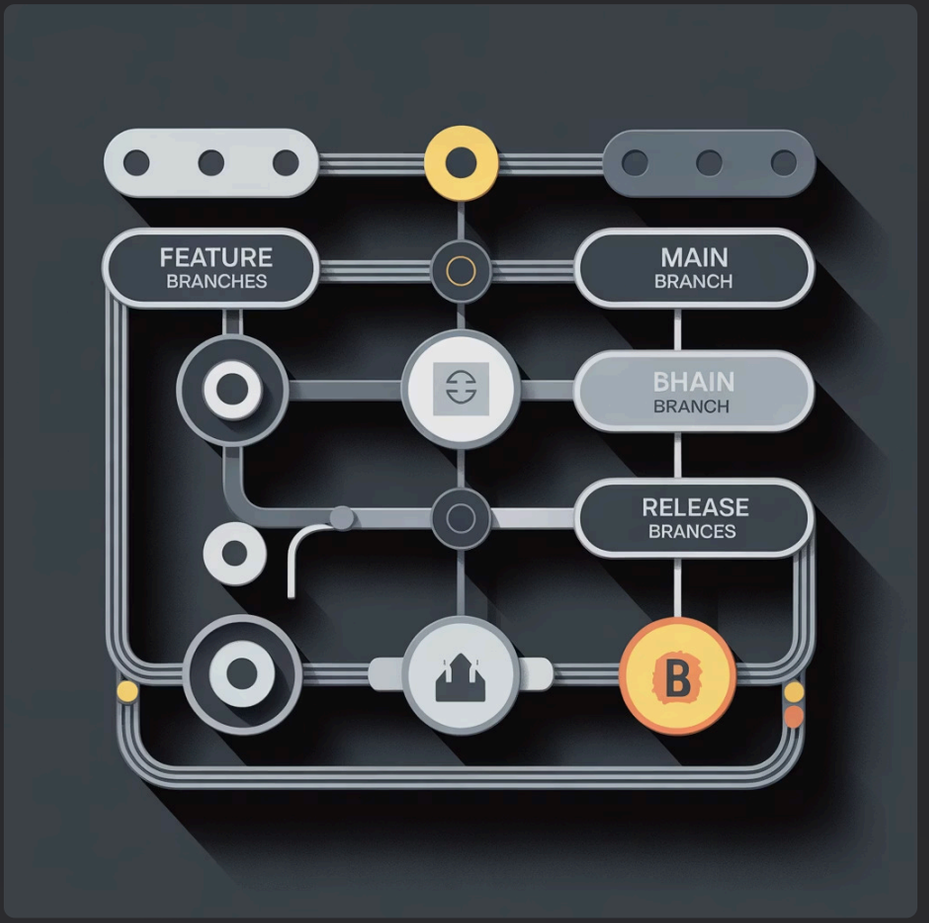
Implementing versioning and tagging strategies from the initial API import prevents technical debt and simplifies long-term maintenance. These organizational mechanisms become increasingly critical as your API portfolio grows.

Versioning strategies to implement immediately:

- **Path-based versioning:** `/v1/resources`, `/v2/resources` - most visible and RESTful
- **Header-based versioning:** `api-version: 2024-01-15` - cleaner URLs, requires client sophistication
- **Query parameter:** `?api-version=1.0` - simple but pollutes query space

Tag applications for API organization:

- Business domain (finance, customer, inventory)
- Lifecycle stage (preview, stable, deprecated)
- Access level (internal, partner, public)



APIM supports both "versions" (breaking changes) and "revisions" (non-breaking changes) - use both for complete change management.

Consumer-Friendly Operation Naming

Operation names and descriptions in APIM serve as the primary interface for developers discovering and understanding your APIs. Well-crafted operation metadata significantly reduces integration time and support burden while improving API adoption.

Use Clear, Action-Oriented Names

Replace auto-generated names like `GetUserById` with `Get user details`. Use natural language that describes the business outcome, not just the technical operation.

Provide Comprehensive Descriptions

Include what the operation does, required parameters, expected responses, and common error scenarios. Document rate limits, data freshness, and performance characteristics.

Align with Consumer Mental Models

Structure operations around user tasks and business processes, not internal system architecture. Group related operations logically using tags.

Include Rich Examples

Provide request and response examples for each operation. Use realistic data that demonstrates common use cases and edge conditions.

Key Takeaways: Backends & API Importing

Successful APIM implementations begin with thoughtful backend configuration and import strategies. These foundational decisions affect long-term maintainability, security posture, and developer experience.

Choose the Right Import Method

Match import approach to your API maturity and existing documentation. OpenAPI import for documented APIs, manual creation for new designs, Azure resource import for native services.

Secure Backend Configuration

Leverage managed identities and Key Vault integration for secrets. Never hardcode credentials in policies. Implement certificate-based authentication for high-security scenarios.

Plan for Scale from Day One

Use named values for environment-specific configuration. Implement versioning and tagging before they're needed. Design consumer-friendly operation names and descriptions.

Maintain Stable Foundations

Keep backend base URLs stable through DNS abstraction and named values. Separate infrastructure concerns from API definitions. Enable smooth environment promotions.

Lab

Thank You for Attending!