Module 1.2: Creating an APIM Instance & Core Concepts

Building a strong foundation in Azure API Management requires understanding its core architecture, entities, and management approaches. This module equips you with the essential knowledge to confidently create, configure, and manage APIM instances across your development lifecycle.

Module Overview

What We'll Cover Today

01

APIM Resource Creation

Regions, SKUs, and deployment options for your infrastructure

02

Core Entities & Terminology

APIs, operations, products, subscriptions, backends, and more

03

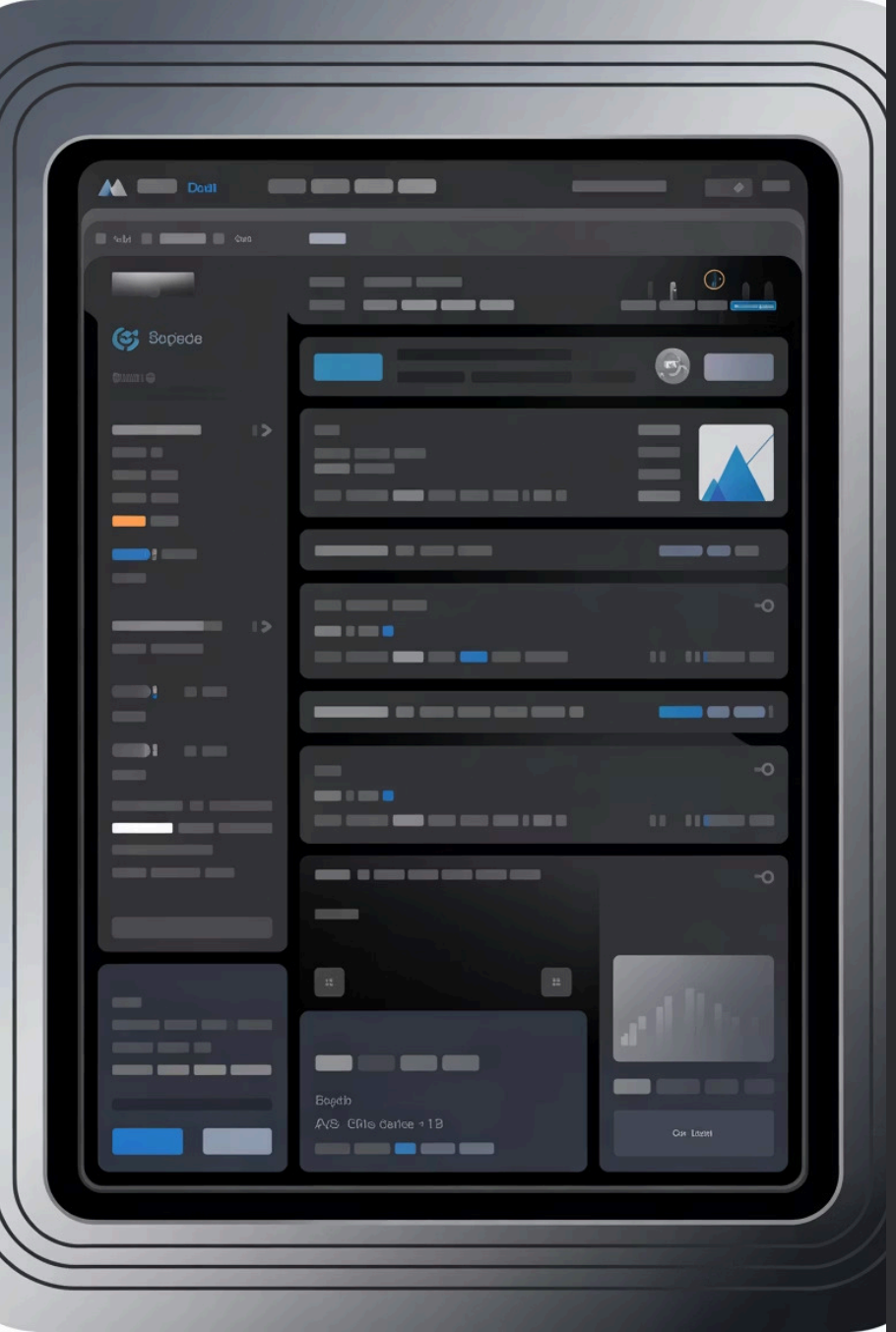
Management Approaches

Portal, ARM, Bicep, CLI, REST API, and Terraform options

04

Lifecycle Strategies

Dev/test/prod patterns and promotion workflows



Creating an APIM Instance

Resource Creation Essentials

When provisioning a new APIM instance, you'll make several critical decisions that impact performance, cost, and architecture. The creation process involves selecting your Azure subscription, resource group, region, and instance name.

The region choice affects latency for your API consumers and determines data residency compliance. Instance names must be globally unique and become part of your default gateway URL.

Key Decisions

- Azure subscription and resource group
- Geographic region selection
- Globally unique instance name
- Pricing tier (SKU selection)
- Administrator email notifications

Understanding APIM SKUs

Developer

No SLA, perfect for development and testing scenarios. Cannot be scaled and lacks production-grade features. Includes all standard features for learning and prototyping.

Basic

Entry-level production tier with 99.95% SLA. Supports up to 2 scale units. Ideal for smaller workloads with moderate traffic requirements and limited custom domain needs.

Standard

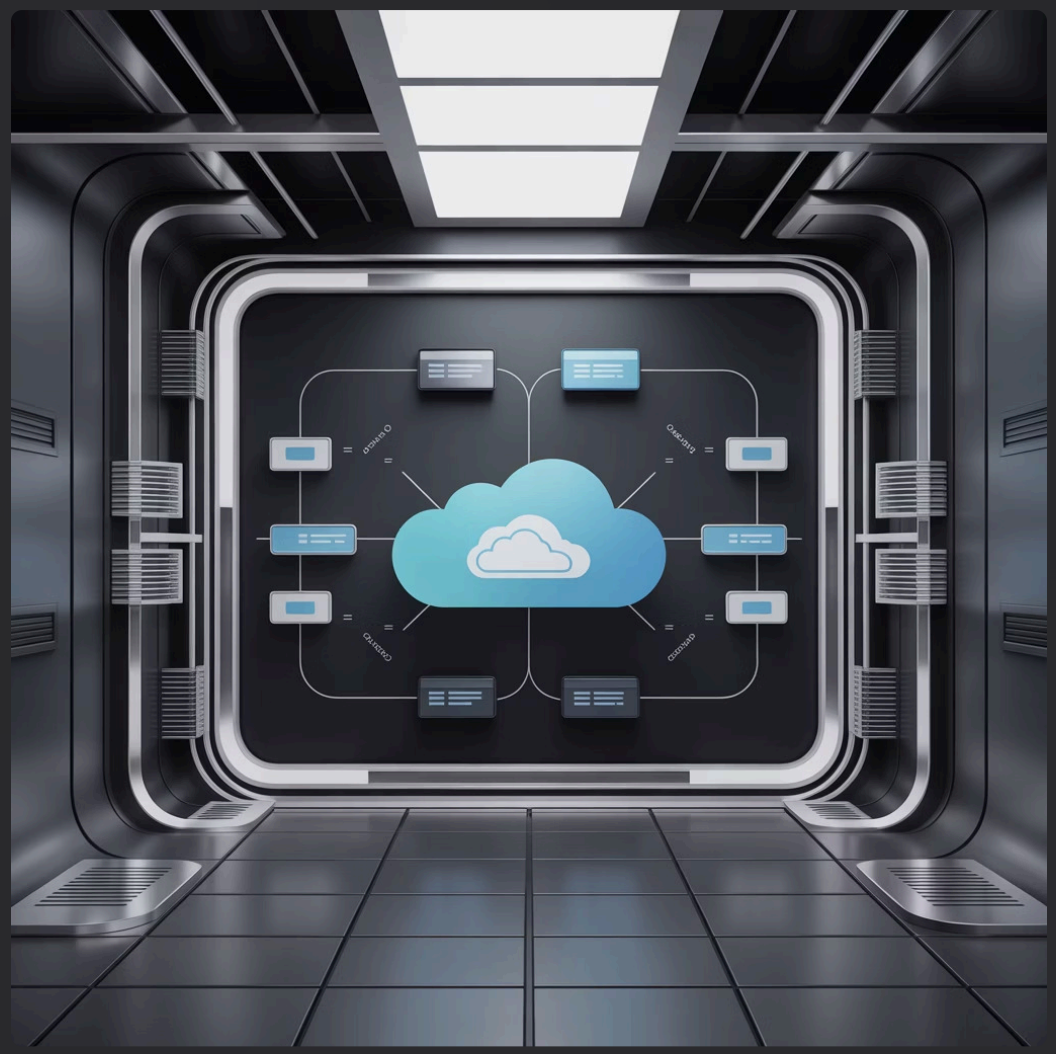
Full production capabilities with 99.95% SLA and up to 4 scale units. Includes multi-region deployment, virtual network integration, and custom domain support for growing enterprises.

Premium

Enterprise-grade tier with 99.99% SLA and unlimited scale units. Supports multi-region deployments, virtual network integration, Azure Cache for Redis, and advanced security features.

Choosing the right SKU involves balancing cost, performance requirements, and feature needs. You can upgrade tiers but cannot downgrade without recreating the instance.

Gateway Architecture: Managed vs Self-Hosted



Managed Gateway

The default option runs entirely within Azure's infrastructure. Microsoft handles all maintenance, updates, and scaling. Your APIs are exposed through Azure-hosted endpoints with built-in high availability.

This option provides the fastest time to value and requires minimal infrastructure management. It's ideal for cloud-native applications and services already running in Azure.

Self-Hosted Gateway

A containerized version of the gateway you deploy to your own infrastructure—on-premises, other clouds, or edge locations. It maintains connectivity to the Azure-hosted control plane for policy synchronization.

Self-hosted gateways enable hybrid scenarios, reduce latency for geographically distributed backends, and meet data residency requirements without sacrificing centralized management.

Core APIM Entities

Understanding the relationship between APIM's core entities is fundamental to effective API management. These building blocks work together to create a flexible, secure, and scalable API infrastructure.

APIs & Operations

The foundation of your APIM configuration

APIs: Your Frontend Contract

What is an API in APIM?

An API in APIM represents the frontend interface—the contract you expose to consumers. It's a logical grouping of operations that abstracts and potentially transforms calls to one or more backend services.

APIs define the path structure, protocols (HTTP/HTTPS, WebSocket), and authentication requirements. They act as the facade between your consumers and backend implementations, enabling versioning, policy application, and traffic management.

Key Characteristics

- Display name and description for documentation
- Base URL path (e.g., `/api/v1/customers`)
- Supported protocols and formats
- Version and revision management
- Associated products and subscriptions

Frontend vs Backend: The API you define in APIM is the frontend contract. It can differ significantly from your backend service structure, allowing you to reshape, combine, or split backend calls.

Operations: The Building Blocks

GET /customers

Retrieve a list of customers with optional filtering, pagination, and sorting parameters. Returns JSON array of customer objects.

POST /customers

Create a new customer record. Accepts JSON payload with customer details. Returns created customer with assigned ID.

PUT /customers/{id}

Update an existing customer by ID. Requires complete customer object in request body. Returns updated customer details.

DELETE /customers/{id}

Remove a customer record by ID. Validates dependencies before deletion. Returns 204 No Content on success.

Each operation represents a specific HTTP method and URL pattern combination. Operations are where you define request/response schemas, query parameters, headers, and operation-specific policies that transform or validate traffic.

Products: Packaging APIs for Consumers

Products are the primary mechanism for organizing and controlling API access in APIM. They bundle one or more APIs together and define the access rules, usage quotas, and terms of use that apply to all included APIs.

Access Control

Products can be **open** (no subscription required) or **protected** (requires approved subscription). Protected products enforce subscription key validation before allowing API calls.

Usage Policies

Apply rate limiting, quotas, and throttling at the product level. These policies automatically affect all APIs in the product, ensuring consistent protection across your API portfolio.

Documentation

Products include terms of use, descriptions, and can be published to the developer portal. This creates a self-service experience where developers discover and subscribe to your APIs.

Subscriptions & Keys: Access Management

The Subscription Model

Subscriptions are the primary access control mechanism in APIM. When a developer wants to call a protected API, they must first obtain a subscription to a product containing that API.

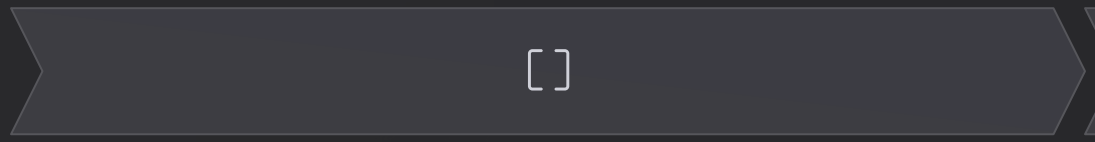
Each subscription generates two keys—primary and secondary—enabling key rotation without service disruption. Keys are passed via the `Ocp-Apim-Subscription-Key` header or as a query parameter.

Subscription Scopes

- **Product Scope:** Access to all APIs in a specific product
- **All APIs Scope:** Master key accessing all APIs (use cautiously)
- **Single API Scope:** Granular access to one specific API

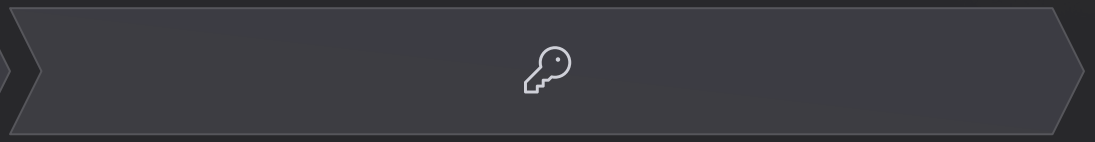
Best Practice: Use product-scoped subscriptions for most scenarios. Reserve all-APIs subscriptions for internal services only, and implement additional authentication layers for production.

Subscription Key Flow



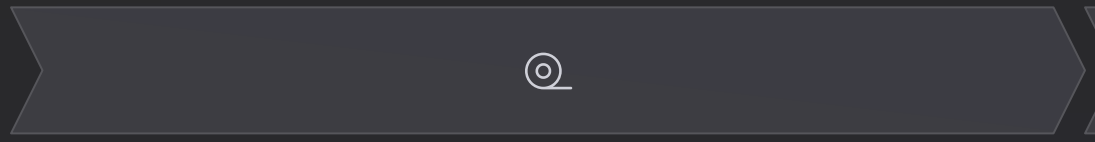
Developer Requests

User discovers API in portal and subscribes to product



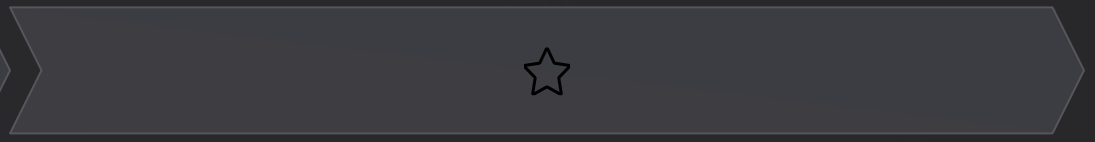
Keys Generated

Primary and secondary subscription keys are created



API Called

Request includes subscription key in header or query string



APIM Validates

Key is verified, policies applied, request forwarded to backend

This flow ensures controlled access while providing developers with immediate, self-service API consumption capabilities. Keys can be regenerated, suspended, or deleted through the portal or API.

Backends & Configuration

Connecting APIM to your services

Backends: Separating Frontend from Implementation

Backends represent the actual services that fulfill API requests. A critical APIM concept is the separation between your **frontend API definition** (what consumers see) and your **backend services** (where requests are routed).

Why Separate Frontend and Backend?

- Version your frontend independently from backend changes
- Route one frontend API to multiple backend services
- Transform requests and responses between formats
- Implement facade patterns and API composition
- Migrate backends without impacting consumers

Backend URL Configuration

Backends can be defined inline in each API or as reusable named backend entities. Named backends support advanced features like circuit breakers, load balancing across multiple instances, and connection pool management.

Backend URLs support Azure services, external HTTP endpoints, Service Fabric, and Logic Apps.

Named Values: Configuration Management

Configuration Storage

Store reusable values like backend URLs, API keys, connection strings, and feature flags. Named values can be plain text or marked as secret for sensitive data.

Key Vault Integration

Reference Azure Key Vault secrets directly as named values. APIM automatically retrieves current values, enabling external secret rotation without APIM configuration changes.

Policy References

Use named values in policies with `{{value-name}}` syntax. This enables environment-specific configurations and promotes policy reusability across APIs.

```
<set-backend-service
  base-url="{{backend-url}}" />

<set-header name="X-API-Key">
  <value>{{external-api-key}}</value>
</set-header>
```

Named values centralize configuration, making it easier to manage environment-specific settings and rotate secrets without modifying policies directly.

Certificates: Securing Backend Communication

Certificate Management

APIM supports uploading certificates for mutual TLS authentication with backends and for custom domain configuration. Certificates can be stored directly in APIM or referenced from Azure Key Vault.

Use client certificates to authenticate APIM to backend services that require certificate-based authentication. This is common in enterprise scenarios and when calling legacy systems that don't support modern OAuth flows.

Common Use Cases

- Mutual TLS (mTLS) authentication to backends
- Custom domain SSL/TLS certificates
- Service-to-service authentication
- Validation of client certificates in inbound policies

Security Tip: Store certificates in Key Vault and grant APIM managed identity access. This enables automatic certificate renewal and centralized certificate lifecycle management.

Management Approaches

Multiple ways to configure and automate APIM

Management Planes Overview

Azure API Management can be configured through multiple interfaces, each suited for different scenarios and workflows. Understanding when to use each approach is key to efficient APIM management.

Azure Portal

The graphical interface for manual configuration, testing, and exploration. Ideal for learning, ad-hoc changes, and quick validation. Provides immediate visual feedback and built-in testing tools.

ARM & Bicep

Infrastructure-as-Code templates for declarative deployments. Best for repeatable, version-controlled infrastructure provisioning. Bicep offers cleaner syntax than raw ARM templates.

Azure CLI

Command-line interface for scripting and automation. Perfect for CI/CD pipelines, batch operations, and shell script integration. Supports both interactive and scripted workflows.

REST API

Direct programmatic access to all APIM capabilities. Enables custom tooling, advanced automation, and integration with non-Azure systems. Most flexible but requires more development effort.

Terraform

Third-party IaC tool with strong multi-cloud support. Preferred in organizations with existing Terraform workflows. Offers state management and provider ecosystem benefits.

Azure Portal: Interactive Management

When to Use the Portal

- **Learning & Exploration:** Discover APIM features and understand configuration options
- **Testing & Debugging:** Use built-in test console to validate API behavior
- **Quick Changes:** Make urgent production fixes or one-off adjustments
- **Visual Configuration:** Design complex policies with syntax highlighting and validation

Portal Limitations

While convenient, the portal has drawbacks for production workflows. Manual changes aren't version controlled, can't be easily replicated across environments, and lack audit trails beyond Azure Activity Log.

Consider the portal a tool for development and emergency fixes, but rely on Infrastructure-as-Code for production deployments.

ARM Templates & Bicep: Declarative IaC

ARM Template Structure

Azure Resource Manager (ARM) templates define infrastructure in JSON format. They're native to Azure, support all Azure services on day one, and integrate seamlessly with Azure DevOps and GitHub Actions.

ARM templates are verbose but comprehensive. They provide complete control over resource properties and dependencies.

Bicep: Cleaner Syntax

Bicep is a domain-specific language that compiles to ARM templates. It offers simpler syntax, better IntelliSense support, and improved readability while maintaining ARM's full capabilities.

```
resource apim 'Microsoft.ApiManagement/service@2023-03-01' = {
  name: apimName
  location: location
  sku: {
    name: 'Developer'
    capacity: 1
  }
  properties: {
    publisherEmail: publisherEmail
    publisherName: publisherName
  }
}
```

Both ARM and Bicep support parameter files for environment-specific values, enabling true infrastructure reusability across dev, test, and production environments.

Azure CLI: Command-Line Power

Interactive Shell

Use `az apim` commands interactively to query APIM state, test configurations, and troubleshoot issues. The CLI provides immediate feedback and supports powerful query filtering with JMESPath.

```
az apim api list --resource-group myRG --service-name myAPIM
--query "[].{name:name, path:path}" -o table
```

Automation Scripts

Combine CLI commands in shell scripts for repeatable operations like bulk API imports, backup routines, or configuration synchronization across instances.

```
#!/bin/bash
for env in dev test prod; do
  az apim nv create \
    --resource-group ${env}-rg \
    --service-name ${env}-apim \
    --named-value-id backend-url \
    --value "https://${env}.backend.com"
done
```

The Azure CLI is particularly powerful in CI/CD pipelines where you need conditional logic, error handling, and integration with other tools beyond pure infrastructure deployment.

REST API: Direct Programmatic Access

Management REST API

The Azure API Management REST API provides complete programmatic control. Every portal operation ultimately calls this API, making it the most comprehensive management interface.

Use the REST API when building custom tooling, integrating APIM management into your applications, or performing operations not exposed through other interfaces. Authentication uses Azure AD service principals or managed identities.

Common Scenarios

- Custom developer portal integrations
- Automated subscription provisioning workflows
- Dynamic policy generation based on external systems
- Real-time configuration queries in monitoring tools

Example: Create API

```
PUT https://management.azure.com/
  subscriptions/{subscriptionId}/
  resourceGroups/{resourceGroupName}/
  providers/Microsoft.ApiManagement/
  service/{serviceName}/
  apis/petstore?api-version=2023-03-01

{
  "properties": {
    "displayName": "Petstore API",
    "path": "petstore",
    "protocols": ["https"],
    "serviceUrl": "https://petstore.swagger.io/v2"
  }
}
```

The REST API returns rich error messages and supports both synchronous and long-running asynchronous operations for resource-intensive tasks.

Terraform: Multi-Cloud IaC

Terraform by HashiCorp is a popular Infrastructure-as-Code tool that supports APIM through the Azure Provider. It's particularly valuable in organizations managing resources across multiple cloud providers or with established Terraform workflows.

State Management

Terraform maintains infrastructure state, enabling it to detect drift between declared configuration and actual resources. This makes it easier to manage existing APIM instances created outside Terraform.

Provider Ecosystem

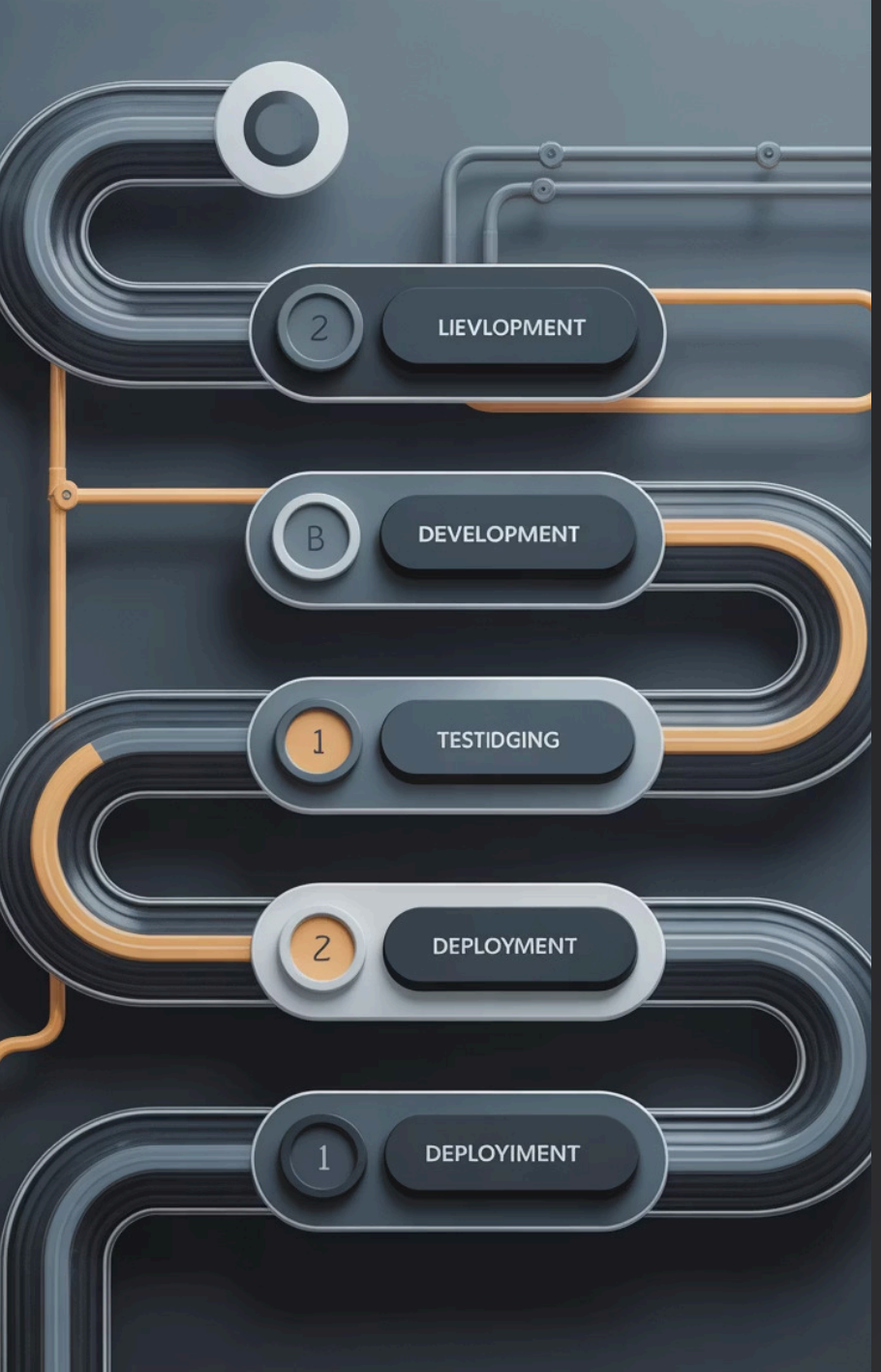
Integrate APIM provisioning with other services using Terraform's extensive provider library. Manage databases, DNS, monitoring, and APIM in unified configuration files.

Plan & Apply Workflow

Preview changes before applying them with `terraform plan`. This reduces the risk of unintended modifications in production environments and provides clear change visibility.

Lifecycle & Promotion

Managing APIM across environments



Development Lifecycle Strategy

Effective APIM management requires a clear strategy for progressing changes from development through testing to production. Multiple approaches exist, each with distinct trade-offs regarding cost, isolation, and operational complexity.

1. **Development**

   Engineers experiment with APIs, policies, and configurations. Frequent changes, relaxed security, and integration with dev backend services.

2. **Testing**

   QA validates functionality, performance, and security. Environment mirrors production configuration with test data and isolated backends.

3. **Staging**

   Final validation before production. Production-equivalent configuration, load testing, and business acceptance testing occur here.

4. **Production**

   Live customer traffic. Changes follow strict approval processes, rollback procedures, and comprehensive monitoring.

APIM Instance Strategies

Separate Instances per Environment

Deploy independent APIM instances for each environment (dev, test, prod). This approach provides maximum isolation, independent scaling, and environment-specific configuration without risk of cross-contamination.

Advantages:

- Complete isolation prevents dev changes from affecting production
- Environment-specific SKUs (Developer for dev/test, Premium for prod)
- Independent disaster recovery and backup strategies
- Simplified RBAC and access control per environment

Disadvantages:

- Higher cost (multiple APIM instances)
- Configuration drift between environments
- More complex promotion workflows

Shared Instance with Versioning

Use a single APIM instance with API versions or revisions to separate environments. Less common due to lack of isolation but reduces infrastructure costs.

Advantages:

- Lower infrastructure costs
- Simplified management (one instance)
- Easier configuration consistency

Disadvantages:

- Risk of dev changes impacting production
- Shared rate limits and quotas
- Complex policy and subscription management
- Limited environment-specific customization

Recommendation: Use separate instances for isolation and safety, especially for production workloads.

Promotion Strategies

Moving APIM configurations from lower environments to production requires careful planning and automation. The goal is to promote tested configurations while adapting environment-specific values.

Export Configuration

Extract API definitions, policies, and settings from source environment using portal, CLI, or REST API

Parameterize

Replace environment-specific values with parameters or named values

Validate

Test configuration in target environment's staging area or using dry-run capabilities

Deploy

Apply configuration to target environment with appropriate approvals and rollback plans

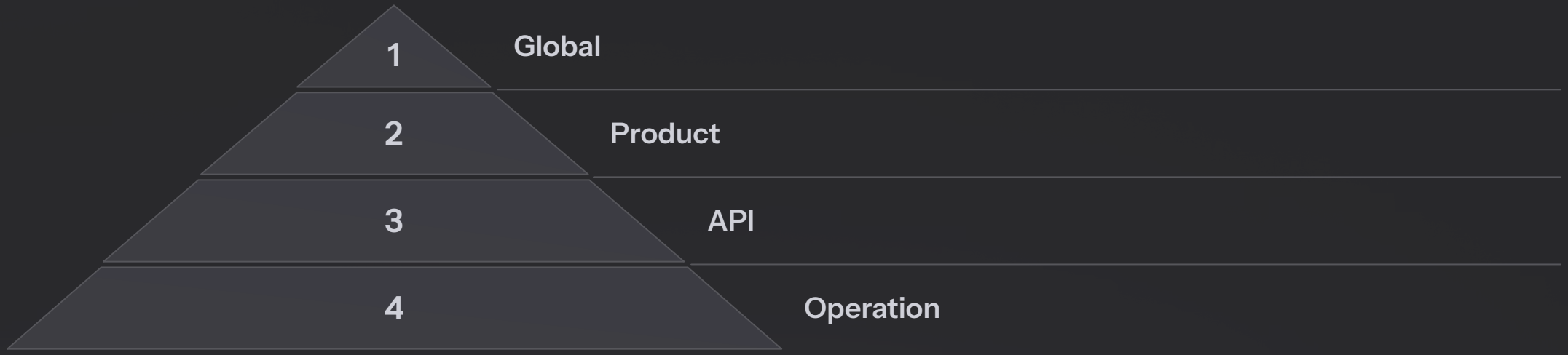
---

Automation Tools

Leverage Azure DevOps pipelines, GitHub Actions, or custom scripts to automate promotion workflows. Store configurations in Git repositories for version control and audit trails. Use ARM templates or Terraform for repeatable, infrastructure-as-code deployments that ensure consistency across environments.

Policy Scopes: Where Policies Apply

Understanding policy scopes is critical for effective APIM configuration. Policies can be applied at four different levels, with each level inheriting and potentially overriding policies from parent scopes.



Policies execute in order from global to operation level. Use `<base />` tags in lower-level policies to explicitly control inheritance. Without `<base />`, parent policies are ignored, allowing complete overrides when necessary.

Policy Scope Best Practices

Global Scope

Apply cross-cutting concerns that affect all APIs: security headers, CORS configuration, logging, and error handling. Global policies ensure consistency and reduce duplication.

Common Global Policies:

- Set security headers (HSTS, X-Content-Type-Options)
- Configure CORS for browser-based clients
- Enable Application Insights logging
- Implement global error handling

Product Scope

Enforce product-level rate limits, quotas, and authentication requirements. These policies apply to all APIs within the product.

Common Product Policies:

- Rate limiting and quota policies
- Subscription key validation
- JWT token validation
- IP filtering for specific product tiers

API Scope

Apply API-specific transformations, backend routing, and caching strategies. This is the most common level for custom logic.

Common API Policies:

- Backend URL rewriting
- Request/response transformation
- Caching configuration
- Mock responses for testing

Operation Scope

Handle operation-specific requirements like parameter validation, response filtering, or unique backend routing for individual endpoints.

Common Operation Policies:

- Parameter validation and transformation
- Operation-specific rate limits
- Conditional backend selection
- Response schema validation

Key Takeaways

Core Entities

Master the relationships between APIs, operations, products, subscriptions, and backends. Understanding these building blocks is fundamental to everything else in APIM.

Frontend vs Backend

Always remember the separation between your API frontend (consumer contract) and backend services (implementation). This abstraction enables versioning, transformation, and migration flexibility.

Automation First

Use Infrastructure-as-Code approaches (ARM, Bicep, Terraform) for production deployments. The portal is for learning and emergencies, not production workflows.

Policy Scopes

Understand where policies apply—global, product, API, and operation levels. Use appropriate scopes to avoid duplication and ensure maintainability.

With these core concepts mastered, you're ready to dive deeper into importing APIs, configuring policies, implementing security, and building production-ready API management solutions.

Lab

Thank You for Attending!