CHAPTER 2

Planning and Managing AI Solutions in Microsoft Azure

Building AI solutions in Azure often feels like constructing a skyscraper; you can't just focus on laying bricks (or writing code). Many an organization learns this the hard way when a perfectly accurate customer sentiment model is rejected because the engineer overlooked compliance checks for European user data. The truth is that successful AI engineering requires orchestrating stakeholders, security protocols, and cost controls as much as it demands technical skill.

Take a retail inventory forecasting tool: while developers obsess over long short-term memory (LSTM) models, warehouse managers care about latency, finance teams demand cost alerts via Microsoft Cost Management, and security leads insist on RBAC roles like "Cognitive Services Data Viewer" to lock down supply chain data. The AI-102 exam tests this kind of big-picture thinking by requiring you to know when to use Azure OpenAI's GPT-4 for creative copywriting and when to use Azure AI services' prebuilt Text Analytics for straightforward sentiment checks—all while avoiding the "$10,000/month cloud bill" horror stories.

Security isn't an afterthought—it's the foundation. Imagine deploying a medical imaging model that accidentally exposes patient IDs due to a misconfigured Azure Kubernetes Service (AKS) cluster. I've seen teams waste months retrofitting security when they could've baked it in up front by using Azure Policy to automatically block noncompliant deployments. And let's talk costs: a logistics company I worked with slashed its image-processing bill by 55% by using Azure Monitor to kill idle inference endpoints—zombie resources that were quietly draining their budgets.

This chapter will arm you with battle-tested strategies to balance performance, security, and costs, whether you're fine-tuning Phi-3 models or integrating Azure

Cognitive Search. The goal? Transform yourself from a coder into an architect who ships AI solutions that survive real-world chaos.

The Azure AI Project Lifecycle

To successfully develop an AI solution on Microsoft Azure, you must understand the different phases of the project lifecycle—each of which is critical to the successful deployment and operation of your solution. This knowledge is what guides developers, data scientists, and other stakeholders from developing an initial concept to realizing a fully operational AI system. In this section, we'll look at the phases of the development lifecycle and explore how to design AI solutions with that cycle in mind. How do you translate vague stakeholder requests ("Make it smart!") into technical specs? When should you pivot from prebuilt AI services to custom models? And why is "deployment" not the finish line but the starting gun for the real work—like catching model drift before it alienates users? I'll help you bridge the gap between textbook lifecycles and the messy reality of shipping AI that doesn't just work but lasts.

As an AI engineer, you must understand the different phases of AI solution development (see Figure 2-1). This is crucial because you will not only develop the solution but also oversee its lifecycle and work with relevant stakeholders to provide insightful, phase-specific recommendations.

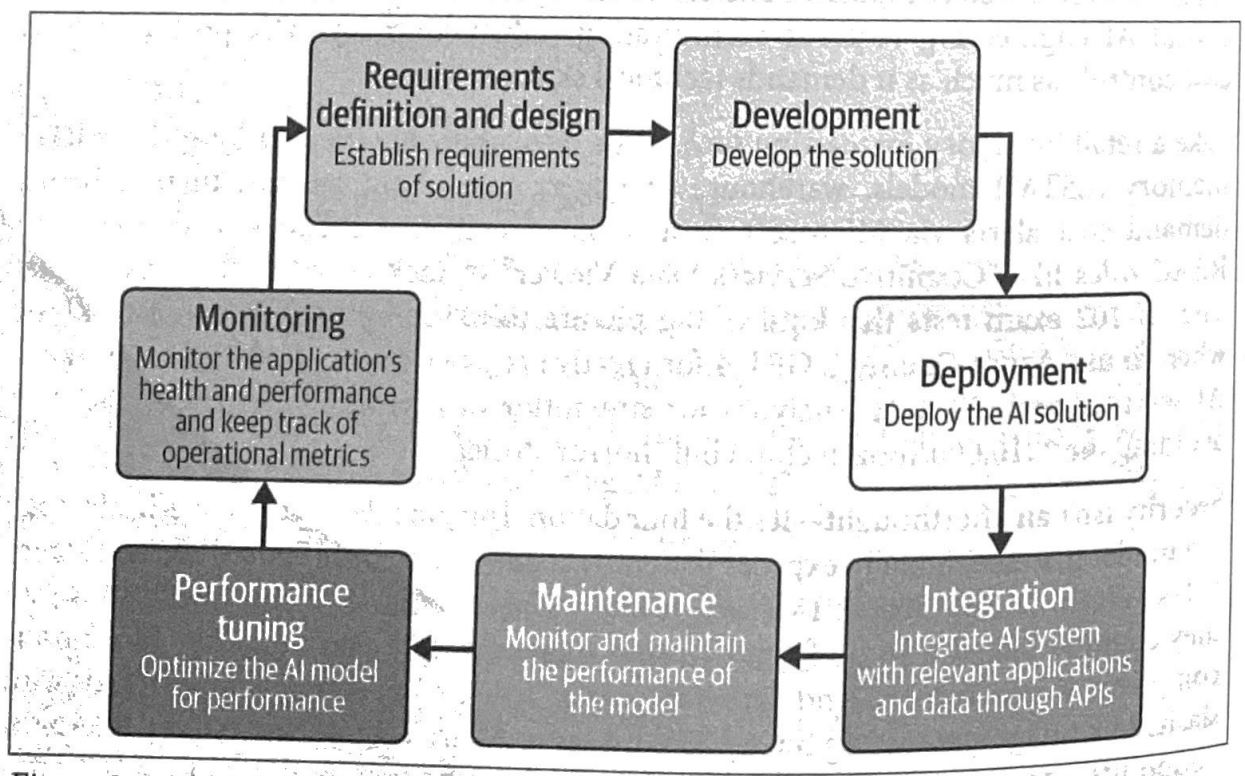

Figure 2-1. The AI development lifecycle

Let's discuss what is expected to take place in each phase.

Requirements definition and design

There are three key steps in the requirements definition and design phase:

- Objective definition
- Conducting a feasibility study
- Resource assessment

First, in the objective definition step, you define the problem that the AI solution aims to solve and specify the desired outcomes and performance metrics. Examples of standard objectives are improving the customer service experience and creating a knowledge base for a help bot.

Then, in the second step, you assess the feasibility of the AI solution in terms of the available data, technology, budget, and time constraints. This ensures that you consider the potential return on investment, which is one of the factors that you'll need to pitch to senior leadership.

Finally, in the third step, you determine which Azure resources and services you'll need to use for the project. This is where you start recommending solutions based on best practices and employing design thinking methodologies to ideate innovative AI solutions. This ensures that you keep the end user at the center of the process, be they an external customer or an internal stakeholder. You'll also look at designing the architecture of the AI solution, which involves considering what AI services to use and integrating them into your environment. For example, you'll make key architectural decisions such as which data storage, compute resources, and monitoring solutions to use.

Beyond these three key steps, as part of designing the solution's architecture, you have to include security every step of the way. You'll need to assure that the design adheres to security best practices and compliance requirements, so you'll want to leverage Azure's ready-to-use security tools or third-party ones as necessary. As discussed in the previous chapter, security is a key component of any AI system, especially given the volume of potentially private data that it handles.

To make these requirements tangible, teams often define success criteria using specific key performance indicators (KPIs). Examples include a target accuracy or precision-recall threshold for a prediction model, or a desired latency of under 200 milliseconds for online inference. Organizations might also measure user satisfaction with AI-driven features via metrics like the Net Promoter Score (NPS) or the average resolution time for support tickets. Common pitfalls during requirements gathering include overly broad objectives, lack of stakeholder alignment on success metrics, and insufficient consideration of data availability and quality. Ensuring clear and measurable KPIs during this phase also helps prevent confusion in later phases.

Development

In the development phase, you will start the process of working on your AI solution. Your first step will be to prepare your data for use. This will include collecting, cleaning, and preprocessing the data accordingly, as well as using tools such as Azure Databricks for data engineering tasks as required.

Then, you will need to develop the AI models themselves. Based on what you accomplished in the requirements definition and design phase, you may or may not already have a specific model and approach in mind for implementing the solution. If you don't, you can experiment with different algorithms, features, and hyperparameters to figure out which model will perform best. This approach should be supported by continuous testing of the models for accuracy, performance, and bias. To assist with this process, you can leverage the capabilities of services such as Azure Machine Learning, which offers tools for model testing and validation, to support this process.

Deployment

In the deployment phase, you will deploy the AI models you've developed into production. You'll need to choose the appropriate deployment target, such as an Azure Virtual Machine (VM), AKS, or Azure Container Instances (ACI), based on the scale and your requirements. Before deciding on a deployment strategy, make sure that you have scoped out the system's performance needs and understand benefits and drawbacks of each option.

This is also the phase where you set up continuous integration and continuous deployment (CI/CD) pipelines, using Azure DevOps or another relevant tool, to support automated testing and deployment. This will help you standardize and streamline how you provision your solutions and ensure that they all undergo the same checks for functionality and performance against set metrics. It will also make it easier to refine the models and reference the performance of previous versions at a later stage.

Organizations are increasingly adopting infrastructure as code (IaC) practices to standardize resource creation and configuration. Tools like Azure Resource Manager (ARM) templates, Bicep, and Terraform allow you to declaratively define the infrastructure (e.g., VMs, AKS clusters) you need for AI workloads. When combined with a YAML-based Azure DevOps or GitHub Actions pipeline, both code and infrastructure can be versioned together, ensuring consistent environments and simplifying rollback if a deployment fails. For example, you could include steps in a CI/CD pipeline to provision or update infrastructure via Terraform scripts and then deploy the latest AI model container image to that infrastructure, thus keeping model versioning aligned with application code releases.

Integration

In the integration phase, you'll create the necessary APIs for your AI models using services such as Azure Functions and Azure App Service. This facilitates integration with other relevant applications or services and supports future expansion and usage of the AI systems that you're developing—an essential aspect of developing scalable, maintainable systems.

Integration also extends to the data you're utilizing. This includes streamlining connections to existing databases, applications, and systems through tools like Azure Logic Apps, Azure API Management, or direct integration via SDKs and APIs. This is necessary to ensure that you handle and process data properly, addressing privacy and security considerations and complying with relevant regulations.

Maintenance

In the maintenance phase, your responsibility is to manage and monitor model versions and updates using your chosen AI service's model management capabilities. For instance, Azure ML has built-in tools to manage versioning and apply updates across the systems managed by Azure. You'll also need to monitor the model continuously for potential data drift and performance degradation.

This responsibility extends beyond Azure services to include all custom software that supports your AI workloads and their dependencies. You'll need to keep all software dependencies, such as libraries and frameworks used in the AI solution, up to date. Services like Azure DevOps can help you manage such dependencies while automating updates (including identifying and resolving bugs in your AI solution and applying necessary security patches).

Performance Tuning

Optimization is the key focus of the performance tuning phase. This includes fine-tuning computer resources, storage resources, and scalability features. If you're deploying AI solutions without prior experience, you'll likely need to experiment with different parameters and solutions.

Central to this phase is *model optimization*, where you continuously adjust and refine your AI models to improve their performance. For example, this could involve tuning machine learning models using Azure ML's automated machine learning (AutoML) and hyperparameter optimization features. You can also support model optimization with other services' capabilities, such as by implementing autoscaling for services that may require it or implementing caching for frequently accessed data to improve response times for AI applications.

Monitoring

You need to continuously monitor your AI solution's performance, usage, and health to guarantee that it's operating in line with the required standards. Effective monitoring can be achieved through a variety of tools, both within and outside of Azure. For example, Azure Monitor can help you collect, analyze, and act on telemetry data from cloud and on-premises environments. Third-party solutions such as Datadog can also help you analyze relevant telemetry data and optimize the performance of the systems you have running.

The monitoring phase also involves gathering the right insights. Services like Application Insights can help you detect anomalies or visualize performance, and tools like Log Analytics allow you to collect and analyze logs generated by your resources. This will help you get a clear understanding of performance issues and highlight improvements you can make to get your solutions working the way you want them to. Later in this chapter, we'll explore different monitoring solutions that can be integrated with AI systems.

You need to go through all seven phases of the project lifecycle to ensure that you fully understand the needs of your solution and implement it methodically. Keep in mind that this is an iterative loop of feedback and improvement.

With that, we can now move on to discussing how to design AI solutions for implementation and gain an understanding of the considerations involved.

Practical: Designing an AI Solution

To successfully design and implement an AI solution system, you need to adhere to the following guidelines for conducting each phase of the project lifecycle:

Requirements definition and design

Before writing any code or spinning up services, you need a clear picture of what the solution must achieve and how it will fit into your organization's ecosystem:

- Clearly articulate the business problem or opportunity the AI solution will address.
- Identify and engage stakeholders to gather requirements and define success metrics for the solution.
- Design the initial architecture of the AI solution, considering scalability, data flow, and integration points with existing systems.

Development

With a solid design in hand, the development phase focuses on preparing your data and building a working prototype that stakeholders can review:

- Collect, clean, and preprocess the necessary data, ensuring that it aligns with the defined requirements.
- Select appropriate Azure services, such as Azure ML for custom models or Azure Cognitive Services for prebuilt AI capabilities.
- Develop a prototype of the AI solution, incorporating iterative feedback from stakeholders to refine the models and their integration.

Deployment

Getting your model into production requires both packaging it for the target environment and automating the release process:

- Prepare the AI models for deployment, considering the target environment (e.g., Azure Kubernetes Service for scalability).
- Implement CI/CD pipelines by using Azure DevOps to automate the deployment process.
- Verify adherence to security best practices, such as managing access control through roles with RBAC and protecting sensitive data.

Integration

Once deployed, the AI services must communicate seamlessly with your existing applications and workflows:

- Develop APIs or use Azure Functions to enable seamless integration of the AI solution with existing systems and applications.
- Test the integration thoroughly to establish that data flows correctly and make sure the AI solution functions as expected within the broader ecosystem.

Maintenance

After go-live, you'll need processes in place to keep everything running smoothly and up to date:

- Establish procedures for ongoing monitoring of the AI solution to promptly detect and address issues.
- Plan for regular updates and maintenance of the AI models and the surrounding infrastructure to provide continued performance and security.

Performance tuning

To ensure your solution remains efficient and cost-effective, continuously analyze and optimize its operation:

- Monitor the performance of the AI solution to identify any bottlenecks or inefficiencies.

- Optimize resource usage in Azure to balance performance with cost, potentially leveraging autoscaling features to adjust to varying loads.

Monitoring

Robust observability is key to spotting problems before they impact users:

- Utilize Azure Monitor, Application Insights, or other tools in the Azure ecosystem or from third-party providers to track the performance, usage, and health of the AI solution.
- Set up alerts for critical metrics to ensure that any potential issues are addressed proactively.

A Simple Example of Solution Design

This example walks through how each phase of the project lifecycle can be applied to solve a real-world problem:

Requirements definition and design

Define the goal to improve inventory management and reduce stockouts and overstock situations through better demand forecasting.

Development

Use historical sales data, combined with external factors like holidays and promotions, to develop forecasting models. Then, leverage Azure ML for model development and experimentation.

Deployment

Deploy the models to Azure Kubernetes Service for scalability and manageability, using Azure DevOps for CI/CD workflows.

Integration

Integrate the forecasting output into the inventory management system through APIs you've developed with Azure Functions, ensuring seamless data exchange.

Maintenance

Establish a schedule for periodic reevaluation and retraining of models with new data to maintain forecasting accuracy.

Performance tuning

Monitor the system's resource usage and adjust the scaling settings in Azure Kubernetes Service to optimize costs and performance.

Monitoring

Implement comprehensive monitoring using Azure Monitor and Application Insights, focusing on model performance metrics and system health indicators, and set up alerts for any anomalies detected.

Weighing the Trade-offs

When architecting such a solution, organizations often weigh trade-offs among cost, performance, and maintainability. For example, teams might choose Azure Functions for simpler event-driven workloads but opt for AKS if container orchestration and scaling are critical. In this case, a service like Azure Container Apps could be a middle ground. Using decision trees or flowcharts can also help teams narrow down which service to use based on factors like concurrency requirements, data volume, and compliance regulations. In real-world scenarios, larger teams might use AKS if they predict large spikes in traffic or want granular control over container configurations, while smaller teams might prefer Azure Web Apps for ease of management and reduced overhead. By evaluating these trade-offs early, teams can ensure that their solutions align with business objectives and technical constraints.

Note that the sample solution design process detailed here is one of many options. There's no one right way to implement the workloads that you require, and there may be alternatives that align more closely with your best practices. We'll discuss this further throughout this book so that you can make informed decisions.

Now, let's look at how to plan and configure access and security when developing AI systems.

Planning and Configuring Access and Security

It's impossible to overstress the importance of access and security. This section will help you gain an understanding of the relevant Azure services and features that will allow you to manage and protect your AI resources.

Implementing the Appropriate Access Control Requirements

Being able to implement the appropriate access controls for AI services on Azure is a critical part of ensuring the security and privacy of the AI systems that you will build. To be successful on the AI-102 exam, you must understand key aspects of data protection, managing account keys, working with the Azure Key Vault, and managing private communications. These components all play pivotal roles in securing AI services and data within the Azure platform.

Data protection

Data protection is at the forefront of controlling access to AI services. It involves securing data at rest, in transit, and during processing. Azure provides several ways to protect your data—for instance, it will automatically encrypt data at rest using industry-standard protocols. However, there may be further compliance regulations that you need to abide by, so you can also choose to manage encryption keys yourself or use Azure Key Vault. You'll need to explore your own protection needs for data at

rest, including the relevant regulatory or compliance requirements and the level of access control and encryption that you'll need.

When you're working with AI services, you must ensure that all your data is processed in a secure environment as well. For example, Azure confidential computing allows for hardware-based isolation when you're processing sensitive data.

In addition, when you're working with AI workloads that involve individuals' private data, you need to take extra care to limit who can see that information and protect it from being exposed. You may want to consider using data masking or anonymization techniques to protect individuals' identities, especially when you use such data for training models. Information exposure has become a more salient risk with the advent of generative AI, given the large amounts of data used to train such systems and the number of people who have inadvertently shared private data with them.

RBAC on Azure

The principle of *least privilege* holds that users, applications, and services should receive only the minimum permissions they need to perform their tasks. This reduces the attack surface and limits the impact of compromised credentials.

Implementing least privilege is a best practice in the industry. A key aspect of this is configuring Azure RBAC so that each user or service principal only has the permissions they actually need. For example, you might grant the built-in "Cognitive Services Contributor" role to data scientists so they can manage and deploy models, while assigning the "Cognitive Services Reader" role to business analysts who only need to view model outputs. What follows is a simplified example of how to assign the "Cognitive Services Contributor" role to a specific user at the resource group level via the Azure CLI:

```
az role assignment create
  --assignee useremail@testcompany.com
  --role "Cognitive Services Contributor"
  --subscription SUBSCRIPTION_ID
  --resource-group RESOURCE_GROUP_NAME
```

Replace the placeholders with your actual subscription, resource group, and user email. You can similarly remove or modify roles to enforce the principle of least privilege. Ensuring separate roles for development, operations, and auditing can help maintain a clear separation of duties.

Managing account keys

When you're dealing with Azure account keys, managing them appropriately is crucial. Account keys must be regularly rotated and updated to minimize the risk of key compromise. Azure provides several mechanisms to automate this. Azure Key Vault is the main service for storing and managing keys; it allows you to configure policies

to automatically rotate keys on a defined schedule or based on specific conditions. Other services, such as Azure App Service and Azure SQL Database, can utilize these rotation policies as well.

It's also recommended that you use Azure resources when working with Azure managed identities. This eliminates the need for you to manage secrets within code, thus reducing the possibility of credential leaks.

Working with Azure Key Vault

Azure Key Vault is a centralized cloud service that is used to store application secrets, encryption keys, and certificates. It's a one-stop shop where you can manage keys and secrets centrally and securely, reducing the risk of secret leakage and simplifying administration.

You can define access policies within Key Vault to control who can access and manage the keys and secrets stored there. You can use Microsoft Entra ID to authenticate and control this access; you'll see an example of this in the practical exercise at the end of this section.

Another advantage of Azure Key Vault is that it enables you to monitor and log all activity within the service. This means you can audit access to secrets and keys and track who accessed what and when, helping you meet security and compliance requirements.

In practice, many organizations set up Key Vault to automatically rotate secrets, keys, and certificates on a regular schedule, such as every 30 to 90 days. Also, if you integrate Key Vault with Azure managed identities, resources like Azure VMs and Azure Web Apps can request secrets or certificates programmatically without storing any credentials in code. The following is a brief example of code that retrieves a secret from Key Vault in Python and includes basic error handling:

```
from azure.identity import ManagedIdentityCredential
from azure.keyvault.secrets import SecretClient

def get_secret_from_key_vault(vault_url, secret_name):
    try:
        credential = ManagedIdentityCredential()
        client = SecretClient(vault_url=vault_url, credential=credential)
        secret = client.get_secret(secret_name)
        return secret.value
    except Exception as e:
        print(f"Error retrieving secret '{secret_name}': {e}")
        return None

vault_url = "https://mykeyvault.vault.azure.net/"
secret_name = "MySensitiveSecret"
secret_value = get_secret_from_key_vault(vault_url, secret_name)
```

```
if secret_value:
    print("Secret retrieved successfully.")
```

Note that if you're not yet logged into your account through the Azure CLI, you may need to use the following command to log in and run the code:

```
az login --scope https://graph.microsoft.com//.default
```

You may also need to ensure that the Key Vault's access configuration is set to use Azure role-based access control and that you have created an access policy that allows list permissions for keys and secrets.

Also note that if a request fails (due to permission issues or a missing secret), the exception will be caught and logged. This kind of error-handling pattern helps ensure that your application can gracefully handle temporary Key Vault issues. For certificate management, you can apply a similar approach: allow Key Vault to automatically renew certificates and simply retrieve them via managed identities within Microsoft Entra ID whenever you need to.

Managing private communications

When you're working with AI, you must ensure that communication between the sender and the receiver of data is private to protect the integrity and confidentiality of that data. There are several strategies that you implement to secure communication.

One of the key methods is using Azure Private Link to access Azure services. This lets you connect through a private endpoint within your virtual network, thus ensuring that data never traverses the public internet. You should also recommend that you always use Transport Layer Security (TLS)/Secure Sockets Layer (SSL) encryption for data in transit. The latest version of TLS is 1.3, but you can ensure proper security by using at least version 1.2. When working with AI services and applications, enforce encrypted connections to protect data as it moves between services and users. This is not only a best practice but often a requirement to comply with industry regulations. Finally, when considering the types of data that can be input into AI systems, assess the associated risks and verify that appropriate end-to-end encryption mechanisms are in place throughout the solution.

Working with Security over the Network

To develop secure solutions, you must work on a network that's secure. This will also help you comply with any regulations and requirements you may be subject to. In this section, we'll discuss some of the security controls that are available and which ones to implement in which scenarios.

Virtual networks and subnets

Azure Virtual Network (VNet) allows you to create logically isolated sections of the Azure cloud in which you can launch Azure resources. *Subnets*, which are segmented portions of an IP network that share a common address prefix, help facilitate efficient network management and security. By dividing a VNet into one or more subnetworks, you can group group and isolate resources based on your specific security and operational needs. Deploying Azure AI services in a VNet allows you to control inbound and outbound traffic, limiting it to only approved resources and minimizing exposure to threats.

Network security groups

Using network security groups (NSGs) is an important part of ensuring that network traffic to and from Azure services (including AI services) is secured. NSGs let you define security rules that allow or deny traffic to resources connected to Azure VNets. By applying NSGs to AI service resources, you can enforce your security policies at the network layer, ensuring that only allowed traffic can access them.

Azure Application Gateway and Web Application Firewall

Azure Application Gateway is a web traffic load balancer that helps manage web applications. When integrated with Azure Web Application Firewall (WAF), it provides a security layer that protects AI services from common web vulnerabilities such as SQL injection, cross-site scripting, and other potential exploits. The WAF will be preconfigured with security rules that you can customize to meet the security requirements of your organization.

Azure Firewall

Azure Firewall is a managed security service that protects Azure VNet resources through threat intelligence-based filtering for inbound and outbound traffic. It's highly available and supports scaling, and because it ensures that all traffic goes through and is filtered by it, it also provides a centralized point for traffic inspection and logging.

Networking for individual AI service instances

Azure AI services are accessible from all networks by default, but you can configure them to restrict access from specific network addresses. This can be done directly within the service's configuration, in the Networking section (see Figure 2-2).

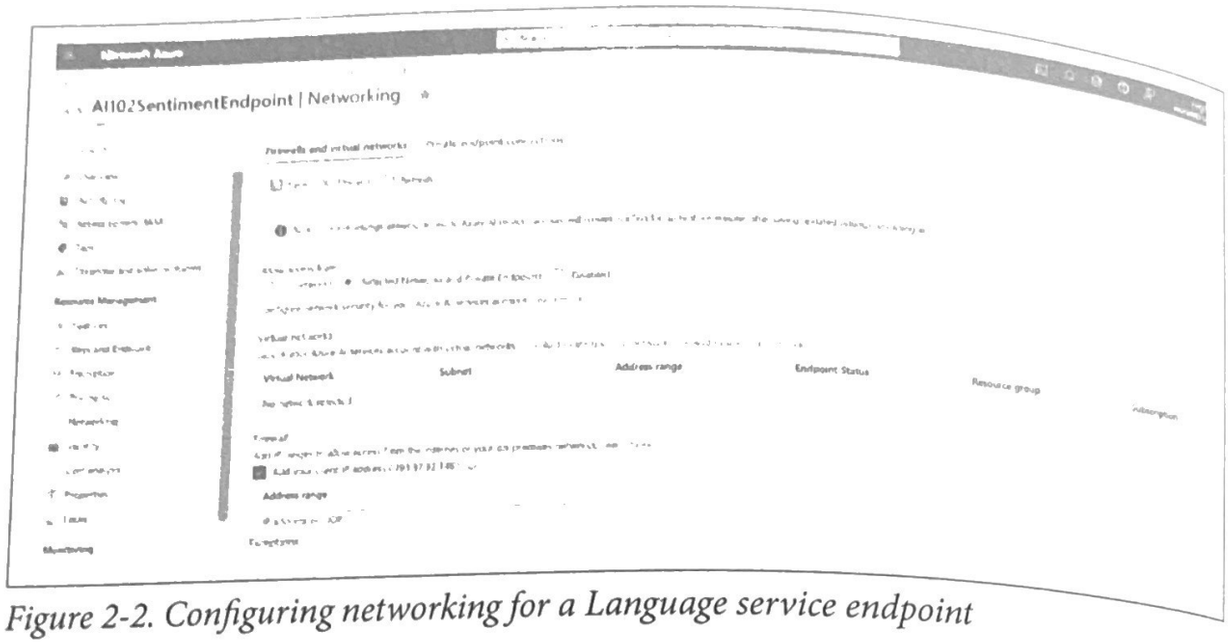

Figure 2-2. Configuring networking for a Language service endpoint

As you can see, you can specify whether all networks or only selected networks and private endpoints are allowed to access the service, or you can disable access completely. When selecting networks, you can choose from existing virtual networks or create new ones if necessary. You can also add an address range that specifies the IP addresses that can access the endpoint. This gives you fine-grained control over who can access your AI service instances and helps you reinforce your overall network security posture.

Now that you've learned about access control and security, we can move on to creating and managing Azure AI services.

Creating and Managing Azure AI Services

When you're designing an AI solution, you should have a clear understanding of how you will deploy the solution and how you and its end users will be able to access it. In this section, we'll discuss how you can deploy an AI resource into a container through CI/CD pipelines, and we'll examine how you can work with APIs and SDKs to access these resources from your own environment.

Deploying an Azure AI Services Resource

First, we'll explore deploying AI services from CI/CD pipelines, which are set up not only to build code but also to deploy it after it passes the necessary tests and goes through the other steps in the pipeline. Let's look at how you can create and deploy such pipelines and how to deploy to a container that meets the requirements of your solution.

Deploying with CI/CD is key to streamlined, repeatable deployment process, so it's important to understand how to architect your pipeline and manage it properly (see Figure 2-3).

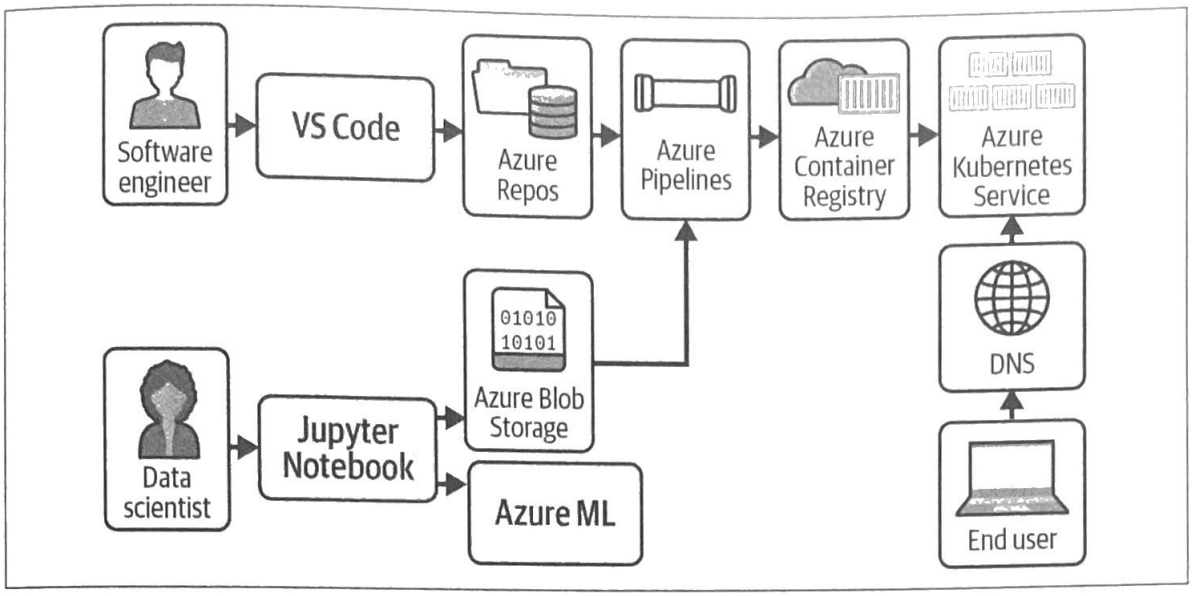

Figure 2-3. A CI/CD pipeline for an Azure AI workload deployment

The first step is to ensure that you have a version control system set up. You can evaluate different options, such as Git or Team Foundation Version Control (TFVC), both of which work natively with Azure Repos. They ensure that every change is tracked and can be rolled back if necessary, which is imperative when you're managing AI models, code, and pipeline configurations.

Once a commit is made to the repository, Azure Pipelines can automatically trigger a build and run tests to validate the performance and accuracy of your AI models. These CI practices ensure that updates are consistently integrated and validated before progressing further in the pipeline. The latest AI model will be pulled from storage and packaged with the application code within a container, and the tests can include benchmark models that can be run against the current model to check whether it meets the required accuracy standards. For example, if you're integrating custom speech services into your application, the CI process will involve training new models from different data sources and testing them to check whether they perform better than previous versions.

Once you've successfully implemented CI, you can begin the CD phase, in which you deploy the improved models. You'll create endpoints for the updated models, thus ensuring that they are available for integration into your solutions. During this process, especially in production environments, the models may need to undergo manual review and approval. This will ensure deployments happen at optimal times and under supervision, in case any issues arise. If the build is approved and passes all the tests, it will be stored as a container image in the Azure Container Registry (ACR).

The container will then be deployed to the appropriate service, such as AKS, making the application available for use.

Thereafter, you must continue monitoring the application's performance and collecting feedback. This will allow you to quickly identify any problems caused by configuration changes so you can revert them or make adjustments. Users will be able to access the AI app's capabilities by connecting to the production endpoint, where it will work just like any other production application.

In later chapters, we'll go through exercises in which you'll deploy AI services in containers and through the CI/CD process. For now, let's look at cost management.

Cost Optimization Strategies for Azure AI Services

Effective cost management for Azure AI starts with securing the right compute capacity and service tiers. For steady workloads, you should reserve Azure VM instances and provision throughput units in Azure OpenAI to lock in discounted rates. Turn on autoscale for your Machine Learning compute clusters so that you pay only for the resources in use. For noncritical training or batch inference jobs, consider using spot virtual machines or low-priority compute to cut costs by as much as 80%.

Beyond compute, you should streamline how you consume AI services. Batch your Cognitive Services calls and resize or split large inputs, like high-resolution images, so you stay within free usage tiers and avoid extra transaction fees. Choose leaner models where they fit, for example, by using embeddings for semantic search or choosing GPT-3.5 Turbo instead of GPT-4 for everyday conversational workloads to keep token charges down. Finally, set up budgets and alerts in Microsoft Cost Management, and review Azure Advisor recommendations on a regular basis. This will help you spot underused resources and unusual spending patterns before they become costly surprises.

Implementing a Container Deployment

When you're working with an AI solution, you'll need to use a hosting setup for the software that you're going to integrate with your AI components. This setup must include the necessary hardware, operating system, and runtime elements. AI services within Azure are provided as cloud offerings, with the software hosted at Azure data centers. But to obtain certain benefits, you can choose to containerize some parts of the services, which will let you make use of the elements within the container.

A *container* is a lightweight, standalone, executable package that includes everything required to run the software. It incorporates the code, runtime, system tools, libraries, and settings.

Some of the advantages containers provide are:

- Improved portability across different platforms, enabling quick updates and rollbacks
- Optimization of DevOps practices through increased workflow adaptability and streamlining of governance and synchronization

To deploy a container in Azure, you pull the image from the ACR, create the container instance, and adjust the necessary configurations. You have a number of deployment options, including a Docker server (on or off premises), an Azure Container Instance, and an AKS cluster.

When deploying an Azure AI service container image, you'll be required to configure three settings: the key and endpoint of the service and the accept value, which you'll have to input to accept the license agreement for the container.

Azure provides container images for a select number of Azure AI services via the Container Registry, which you can use to deploy a specific part of the AI service. Table 2-1 describes some of the available container images that you can pull and use for this purpose.

Table 2-1. Available Azure AI service container images

AI service	Features	Images
Language	Key phrase extraction	Azure AI Services – Text Analytics: Key Phrase Extraction (*https://oreil.ly/8ARjG*)
	Sentiment analysis v3	Azure AI Services – Text Analytics: Sentiment Analysis v3 (*https://oreil.ly/pVqio*)
	Text language detection	Azure AI Services – Text Analytics: Language Detection (*https://oreil.ly/WqIN_*)
	Text analytics for health	Azure AI Services – Text Analytics for Health (*https://oreil.ly/izhoH*)
	Translator	Azure AI Services – Translator: Text Translation (*https://oreil.ly/GJLIf*)
Speech	Speech to text	Azure AI Services – Speech Services: Speech to Text (*https://oreil.ly/tb5-1*)
	Custom speech to text	Azure AI Services – Speech Services: Custom Speech to Text (*https://oreil.ly/HCBDY*)
	Neural text to speech	Azure AI Services – Speech Services: Neural Text to Speech (*https://oreil.ly/mYq8N*)
	Speech language detection	Azure AI Services – Speech Services: Language Detection (*https://oreil.ly/qtc0A*)
Vision	Read OCR	Azure AI Services – Vision: Read (OCR) (*https://oreil.ly/4t2Xs*)
	Spatial analysis	Azure AI Services – Vision: Spatial Analysis (*https://oreil.ly/n7BwG*)
Decision	Anomaly detection	Azure AI Services – Decision: Anomaly Detector (*https://oreil.ly/R3pxQ*)

To download these containers, you can simply run a `docker pull` command. Some might be publicly in preview, in which case you must request access.

In later chapters, I'll take you through some practical exercises on container deployment to get you more familiarized with the process.

Working with APIs and SDKs in Azure

When you're working with languages that don't have official SDKs or are less commonly used with AI services, you'll often work with REST APIs, which expose endpoints to allow developers to perform operations such as submitting data for processing, retrieving results, and managing AI models. This is done through standard HTTP methods like GET, POST, PUT, and DELETE. Let's look at what's required to work with REST APIs in the context of Python.

When you're interacting programmatically with Azure AI services—be it via REST APIs or SDKs—production scenarios often require robust error-handling patterns and retry logic. Services may throttle requests or return transient errors (HTTP 429, 503, etc.) under heavy load, and implementing exponential backoff strategies can help. What follows is a simplified Python snippet showing a retry mechanism with the Text Analytics SDK:

```
import time
from azure.ai.textanalytics import TextAnalyticsClient

def analyze_text_with_retry(client, documents, max_retries=3):
    for attempt in range(max_retries):
        try:
            response = client.analyze_sentiment(documents=documents)
            return response
        except Exception as e:
            if attempt < max_retries - 1:
                print(f"Attempt {attempt+1} failed. Retrying...")
                time.sleep(2 ** attempt)
            else:
                raise e
```

Additionally, for advanced rate-limiting scenarios, you can track usage metrics and the throttling headers that are returned by Azure to dynamically adjust request rates. This ensures that your application handles service constraints gracefully, maintain availability, and avoids unexpected failures or cost spikes.

Standard web service API primary operations

The primary operations of a standard web service API are as follows:

GET
: This retrieves data from a server at the specified resource. For example, you could send a GET request to an API to fetch a list of users.

POST
: This sends data to the server to create a new resource. For example, you could use a POST request to post a user's details to create a new user profile.

PUT
: This updates existing data with new information supplied in the request. Note that using PUT may alter an existing user's details with updated data.

DELETE
: This removes data from the server. For example, you could use a DELETE request to remove a user's profile from the database.

Each operation informs the server of the intended action to be performed on the specified resource and plays a vital role in RESTful API design, which uses HTTP requests to manage resources as part of its architectural style.

Relevant API documentation

You'll need to review the relevant documentation for the programming language and the service you're using. The documentation should provide you with details on what you need to input, including the endpoints and their purposes, the required and optional request parameters, the expected request and response formats (e.g., JSON, XML), and the error codes along with their meanings. Reviewing it will help you fully understand how to configure your service and code your system so that the AI service can correctly interpret your requests.

Construction of the HTTP request

When you call the AI service, you'll need to construct an HTTP request. This requires you to choose the HTTP method, such as GET, POST, PUT, or DELETE, and include the necessary headers, such as `Content-Type` and `Authorization`.

The `Content-Type` header field indicates the media type of the resource being sent to the server (in the case of POST or PUT requests) or the format that the client can accept from the server (in the case of GET requests). It also tells the server how to interpret the data included in the body of the request—for example, as `application/json`, `text/html`, or `multipart/form-data`. The `Authorization` header is used to pass credentials and authenticate the user or entity making the request. It typically carries credentials such as bearer tokens, basic authentication credentials, API keys, or JSON Web Tokens (JWTs), ensuring that the requestor has permission to perform the requested operation. Together, the `Content-Type` and `Authorization` headers play a crucial role in ensuring that HTTP requests carry the appropriate data in a recognized format and the requestor has the right to interact with the specified resources. The `Content-Type` is normally `application/json`, while the `Authorization` is normally your API key or bearer token.

For methods such as POST, you will also need to provide a request body, which is usually a JSON object that contains the data that the AI service will process.

Sending requests

In Python, sending a request typically involves an HTTP client such as the *requests* library. You can also use command-line line tools like curl or dedicated API testing tools such as Postman.

The *requests* library is a popular HTTP client that makes it easy for developers to send HTTP/1.1 requests. It provides methods for using different HTTP verbs and includes built-in support for query parameters, form data, multipart files, and custom headers. The library simplifies the process of interacting with web services and handles the complexities of making requests and processing responses, including managing cookies, sessions, and connection pooling.

On the other hand, Postman is an interactive and user-friendly tool for API development and testing that enables you to construct, share, test, and document API requests through a graphical interface. You can read responses, pass data, and add scripts for testing and validation. Postman supports all the standard HTTP methods and lets developers debug APIs and interact with web services without writing any code. That makes API development much more accessible to those who don't have extensive coding experience. Using tools such as Postman also allows you to quickly test whether endpoints are working, in a way that's generally less strenuous than making an HTTP request directly over Python.

Handling responses

After you send a request, the AI service should respond with an HTTP status code, along with a response body if necessary. Most of the codes are standard, such as 2xx if the resource is successful, 4xx if there's a client error, or 5xx if there's a server error. You can parse the response body to extract information returned by the AI service. This may include using Python to deserialize the response, which will normally be in JSON format.

Deserializing a JSON response in Python involves converting the JSON-formatted string, which is typically received from a web server following an HTTP request, back into a Python data structure. This is a process that's commonly handled by the `json` module, which is part of Python's standard library. When a response is obtained, it's often in the form of a string or a byte stream. Using the `json.loads` method, you can parse this string and transform it into a native Python object, such as a dictionary or a list, depending on the JSON structure. This allows you to access and manipulate the data just as you would with any other native Python object. The *requests* library further simplifies this by providing a `.json` method directly in the response object that

performs the deserialization internally so that you can work with the JSON data immediately.

As part of handling responses, you also need to handle errors appropriately.

Handling errors

It's important for you to know what to do if there's an error. This may include retrying the request, logging the error, or taking corrective actions.

For example, a 400 Bad Request response often includes a detailed message explaining what went wrong (e.g., missing required parameters, incorrect data formats) and prompting the user to correct the request format or data. A 401 Unauthorized status code indicates issues with authentication and thus suggests that the user needs to verify API keys or authentication tokens. A 503 Service Unavailable response might imply that the service is temporarily overloaded or down for maintenance, in which case the user would usually implement retry logic with exponential backoff.

Each error response from Azure AI services is typically accompanied by a JSON body that includes an error code and a message providing additional context. You can deserialize this information to determine the root cause, inform the user, or trigger corrective actions in the application. It's also good practice to log these responses, as they often reveal potential security vulnerabilities that may pop up during the development process and provide a mechanism to directly debug issues.

Now that you have a good understanding of how to work with REST APIs, you can put your knowledge to work by designing an AI solution that uses one.

Practical: Designing an AI Solution with the REST API

In this section, you'll develop a solution that interacts with the Azure AI Language service via the REST API. You'll implement two capabilities—key phrase extraction and language detection—through dedicated functions designed to perform these tasks. You'll then call these functions to execute the respective operations.

Follow these steps:

1. Navigate to your Azure portal.
2. Create a Language service resource.
3. Take note of the keys and endpoint after creating the service.
4. Install the Python *requests* library with the following command:

```
pip install requests
```

5. Now, you can start writing the Python code. Create a new file called *azure_text_analytics.py* and open it in your favorite text editor. Start by importing the relevant libraries:

```
import requests
import json
```

6. Then, set up the API key and endpoint by declaring them accordingly:

```
subscription_key = 'your_key_here'
endpoint = 'your_endpoint_here' + '/text/analytics/v3.1/'
```

7. The API will expect the content to be in JSON format, with specific headers:

```
headers = {
    'Ocp-Apim-Subscription-Key': subscription_key,
    'Content-Type': 'application/json',
    'Accept': 'application/json'
}

body = {
    "documents": [
        {
            "language": "en",
            "id": "1",
            "text": "Hello world. This is a test for Azure AI Language."
        }
    ]
}
```

The JSON payload comprises headers and a body, which serve distinct purposes in an HTTP request. The headers contain metadata such as the Azure-specific authentication key (`Ocp-Apim-Subscription-Key`), a declaration that the payload is in JSON format (`Content-Type`), and an indication that the client expects to receive JSON-formatted responses (`Accept`).

The body contains the actual content for analysis, which is organized under `documents`. This is an array of objects, each of which represents a text document and includes a language specifier (`language`), a unique identifier (`id`), and the text to be processed (`text`). This structure enables services like Azure AI Language to receive and process natural language inputs in a clear, consistent way.

8. Next, you'll declare functions for the two capabilities:

```
def key_phrase_extraction():
    key_phrase_url = endpoint + 'keyPhrases'
    response = requests.post(key_phrase_url, headers=headers, json=body)
    key_phrases = response.json()
    print("\nKey Phrase Extraction:")
    print(json.dumps(key_phrases, indent=4))

def language_detection():
    language_url = endpoint + 'languages'
    response = requests.post(language_url, headers=headers, json=body)
    languages = response.json()
    print("\nLanguage Detection:")
    print(json.dumps(languages, indent=4))
```

The `key_phrase_extraction` function is designed to extract key phrases from text. It constructs a URL by appending `keyPhrases` to a base endpoint, thus indicating that it's targeting a key phrase extraction service. The function then makes a POST request to this URL, passing in a set of headers and a JSON payload (in the body) that contains the text to be analyzed. Upon receiving a response, it extracts the key phrases and prints them out in a formatted manner.

Similarly, the `language_detection` function is aimed at identifying the language of a given text. It constructs a service URL by appending `languages` to the same base endpoint, thus indicating it's targeting a language detection service. It then makes a POST request with the same set of headers and payload. The response, which will contain information about the detected language(s), is then parsed from JSON and printed out in a formatted way.

9. Finally, invoke the two functions you've defined to perform the analysis:

```
key_phrase_extraction()
language_detection()
```

10. Then, you can run the script with the following command:

```
python azure_text_analytics.py
```

11. The results will be returned in JSON format (see Figure 2-4). You'll see that the key phrase extraction results list the significant phrases that have been found in the text. The language detection capability will then confirm the language that the text is written in, along with a confidence score.

```
Key Phrase Extraction:
{
    "documents": [
        {
            "id": "1",
            "keyPhrases": [
                "Azure AI Language",
                "Hello world",
                "test"
            ],
            "warnings": []
        }
    ],
    "errors": [],
    "modelVersion": "2022-10-01"
}

Language Detection:
{
    "documents": [
        {
            "id": "1",
            "detectedLanguage": {
                "name": "English",
                "iso6391Name": "en",
                "confidenceScore": 1.0
            },
            "warnings": []
        }
    ],
    "errors": [],
    "modelVersion": "2023-12-01"
}
```

Figure 2-4. Key phrase extraction and language detection results returned in JSON format

12. Now, replace the two lines of code below the `key_phrases = response.json` line in the script. Here is the updated code to show only the key phrases and the detected language object.:

```
phrases = key_phrases['documents'][0]['keyPhrases']
print("\nKey Phrases:")
print(phrases)
```

Replace the two lines below the `languages = response.json` line with:

```
detected_language = languages['documents'][0]['detectedLanguage']
print("\nDetected Language:")
print(detected_language)
```

13. Run the script again. You should get the output shown in Figure 2-5.

```
Key Phrases:
['Azure AI Language', 'Hello world', 'test']

Detected Language:
{'name': 'English', 'iso6391Name': 'en', 'confidenceScore': 1.0}
```

Figure 2-5. Printed key phrase extraction and language detection results

The output now shows only the key phrases and the detected language object. There's no need to review the entire JSON response, so we've streamlined the logic to focus solely on the essential elements.

With that, you've created your first system that's capable of using an AI service's REST API. While you'll most likely only need to take this approach when working with languages that aren't supported by the Azure SDK, it's still best practice to understand how to use REST APIs and interpret JSON responses. This knowledge will be crucial when debugging issues later on. This example also serves as a solid boilerplate for building additional solutions and experimenting with the different capabilities of the Azure AI Language service.

Now, let's turn our attention to monitoring AI services on Azure.

Monitoring Azure AI Services

Monitoring AI services is a necessary part of keeping up to date with how they are performing and whether additional optimizations are necessary. Azure provides a wide range of tools that can support monitoring services while helping you with troubleshooting and auditing. In this section, I'll introduce the different types of tools that are available and explain how and when to apply them.

Proactively Monitoring Costs

Let's face it: AI projects can burn through budgets faster than a GPU on full blast. Last year, a startup client learned this the hard way when its chatbot's training jobs quietly racked up $12,000 in unplanned Azure costs—all because no one noticed a forgotten experimental VM. Proactive cost monitoring isn't just about spreadsheets; it's about building guardrails that keep innovation from bankrupting you. Here, we'll explore tools that act like financial airbags, cushioning your projects against fiscal face-plants.

Microsoft Cost Management

Think of this as your cloud accountant on caffeine. Microsoft Cost Management doesn't just show you static charts—it lets you set budgets that scream (literally, via SMS/email alerts) when your fine-tuning experiments hit 80% of your monthly limit. One ecommerce team I worked with slashed its monthly spend by 25% after spotting idle GPU clusters in its staging environment that were somehow still chewing through $200 a day. Cost Management's custom dashboards can break down costs by project, so you can finally tell your boss exactly how much that experimental facial recognition API is costing. It's important to continuously and regularly review its reports so you can identify and adjust any resource usage patterns that may benefit from optimization.

Azure Advisor

Azure Advisor provides personalized recommendations on how to optimize your Azure resources based on cost, performance, operational excellence, and security factors. In the case of AI services, this may involve resizing or shutting down instances that are underutilized. Azure Advisor may also recommend cost-effective resource options based on best practices to reduce the overall spending that's required. I once saw it save a hospital $8,000 a month by recommending that its staff delete orphaned storage accounts from old patient data experiments.

Cost Analysis

Azure Cost Analysis can break down your AI service costs by resource, location, and tags. Say you're trying to help a retail chain track its AI costs. Without tags, you're staring at a dollar sign soup. But when you tag shelf-monitoring models as "Inventory" and customer chatbots as "Support," suddenly you're able to see that Inventory models are costing three times more than they need to because they're using premium GPUs for simple object detection. By switching to cheaper CPU instances, your client can save $15,000 a month.

Using Metrics and Alerts

Now that we've exposed the budget killers, let's talk survival tactics. How do you automatically pause training jobs when costs spike? Can alerts trigger Azure Functions to downscale services? That's where operational health metrics become your crystal ball.

Using metrics and alerts is an important part of keeping track of how your services are performing, and auditing them when necessary. Azure's sophisticated monitoring capabilities enable you to proactively oversee AI services and ensure that they operate within the desired thresholds of performance and availability. This is crucial when you're working with these services because it helps you identify issues and mitigate

them sooner rather than later. This section discusses setting up alerts and configuring diagnostic logging as part of troubleshooting and auditing.

Alerts

You can configure alerts based on specific metric thresholds or anomalies, allowing you to respond quickly to potential issues. For instance, you can set up an alert for a high prediction error rate in an Azure ML model, or for a spike in API calls in a Language service. High error rates can signify underlying issues with the application or infrastructure that can potentially lead to suboptimal performance, system downtime, or even a complete service outage, all of which can directly impact user experience and satisfaction. Such issues can also result in data loss or corruption, security vulnerabilities, and additional costs due to increased resource consumption and troubleshooting efforts. On the other hand, a sudden spike in API calls may indicate a surge in user traffic or an unintended loop or bug in the code, which could overwhelm the system and lead to throttling, increased latency, and failure to serve legitimate requests. This can also result in financial implications due to Azure's pay-per-use model, with unexpected spikes potentially incurring significant and unforeseen costs.

The good news is that setting up alerts in Azure Monitor is simple: just select the resource you want to set the alert on and specify the conditions. For example, you can indicate the thresholds that, if exceeded, will trigger an alert, then detail what should happen if the alert is triggered (say, an SMS or email notification being sent out). Additionally, you can organize these alerts by grouping them with tags to ensure that the appropriate services are all addressed in one go.

Beyond creating basic alerts, you can create custom metrics such as domain-specific counters—e.g., number of successful user logins or documents processed per minute—and expose them to Azure Monitor using the Metrics API or Application Insights custom events.

In more advanced alerting scenarios, you can integrate Azure Monitor with external monitoring systems like Splunk or Grafana and generate custom dashboards that will aggregate both Azure and on-premises telemetry. This can help you unify operations data across hybrid environments and provide you with a holistic view of AI performance and usage.

Configuring diagnostic logging

Diagnostic logging within Azure AI services provides a comprehensive view of the operational activities that are being performed. This is vital when you need to troubleshoot and audit service behavior, as many issues aren't visible from the frontend. Having proper logging configurations and practices in place is best practice, to ensure the logs will be there for you to refer to when you need them.

Azure lets you enable diagnostic logging for various AI services and direct logs and metrics to different destinations, such as Azure Monitor Logs, Azure Event Hubs, and Azure Storage. This flexibility supports a wide range of scenarios, from real-time monitoring to archiving logs for compliance. Once you've collected the data, you can use Azure Log Analytics to query and analyze it. This will help you identify patterns, trends, and anomalies and provide insights into how your AI services are functioning.

Diagnostic logs can also strengthen the security posture of AI services. For instance, Azure Security Center can use these logs to detect potential security issues and provide appropriate recommendations to mitigate the risks.

Performance metrics

Performance metrics in Azure are crucial for monitoring AI deployments, offering insights into system efficiency and aiding in identifying potential bottlenecks that could affect performance. These metrics will help you make proactive adjustments to ensure that your AI models operate optimally and deliver reliable outcomes in real-time applications.

The performance metrics that you'll use most often when monitoring AI models include:

Latency
: This is the time it takes to receive a response from an AI service after a request is made. This metric is critical in assessing the responsiveness of a service.

Throughput
: This is the number of requests the service can process within a specific time frame. It indicates the service's capacity to handle workload.

Availability
: This is the proportion of time during which the AI service is operational and accessible. A system must have a high availability metric to be reliable.

Quota usage
: This is a measure of the extent to which provisioned resources have been used within the set quota. You'll need to monitor your service's quota usage to understand its resource consumption and avoid service interruptions.

Request size
: This is a measure of the size of requests sent to the service, with larger requests potentially impacting processing times and throughput.

Data operations

This is a measure of the volume of ongoing data operations. For services involving data transactions, measuring the volume of data operations like read/write actions is key to assessing the performance of backend data stores.

Error metrics

It's also vital to measure error metrics, because they provide you with insights into model accuracy and reveal areas for improvement in predictive performance. By tracking error metrics, you can ensure that your AI models remain robust and aligned with desired outcomes. You'll also put yourself in a better position to make timely interventions to enhance overall service reliability.

The key error metric you'll be tracking is the *error rate*, which is the percentage of requests that result in errors. Tracking the error rate will allow you to monitor the overall health and reliability of your AI service—a high error rate can indicate problems with the AI model, data quality, or service infrastructure.

Model-specific metrics

Model-specific metrics in Azure is essential for monitoring AI deployments because they offer you tailored insights into your models' unique characteristics and performance. Monitoring these specialized metrics allows you to better understand and optimize the behavior of your models and ensure that they meet the desired objectives.

The most critical model-specific metric you'll be monitoring is *model accuracy*. With machine learning models, in particular, tracking metrics such as accuracy, precision, recall, and other relevant performance indicators is essential to ensure they're producing valid and reliable outputs.

Depending on your use case, you may need to monitor a broader set of metrics. However, the performance, error, and model-specific metrics described here are the ones you'll use most often when monitoring AI solutions, and they will provide essential visibility into potential issues.

Practical: Designing Your AI Solution

Now that you have a holistic understanding of security and monitoring considerations, you can embark on a practical design exercise to put it all into practice. Here, you'll create an AI solution that aims to perform entity recognition—using the Language AI service—on a specified sentence. However, the main goals of this exercise are to provide secure access to the system within your Azure environment and to set up monitoring that can quickly identify and address potential issues.

Setting up the key vault

To begin, you need to create a key vault. Your service will be able to fetch the necessary keys from the vault, ensuring secure access to the sensitive variables you'll be using throughout this exercise.

Here are the steps you need to follow when setting up the key vault:

1. Navigate to the Microsoft Azure portal dashboard.
2. Select "Resource groups," and use the resource group you created in Chapter 1. Make a note of its exact name; you'll use it in the following step. (If you haven't created one yet, follow the instructions there to do so now.)
3. To create a key vault, run the following command in the Azure CLI:

   ```
   az keyvault create
     --name KEY_VAULT_NAME
     --resource-group RESOURCE_GROUP_NAME
     --location REGION_NAME
   ```

   Replace *KEY_VAULT_NAME* with the name of your key vault *RESOURCE_GROUP_NAME* with the name of your resource group, and *REGION_NAME* with the region you will be using.
4. Create a service principal by running the following command:

   ```
   az ad sp create-for-rbac --name PRINCIPAL_NAME
   ```

   Replace *PRINCIPAL_NAME* with the name of your principal. Take note of the app ID, password, and tenant ID displayed in the output; you'll use them later to authenticate your application to Azure services.
5. Grant the service principal access to the key vault by running the following command:

   ```
   az keyvault set-policy
     --name KEY_VAULT_NAME
     --spn YOUR_APP_ID
     --secret-permissions get list
   ```

   Replace *KEY_VAULT_NAME* with the name of your key vault and *YOUR_APP_ID* with the app ID that you noted when you ran the `az ad sp create-for-rbac` command.

   If you encounter an error because you can't set policies with `--enable-rbac-authorization` specified, go to the Azure Key Vault resource in the Azure portal, choose Settings → Access Configuration, select "Vault access policy," and click Apply.
6. Store the Azure AI service key in the key vault by running this command:

```
az keyvault secret set \
  --vault-name KEY_VAULT_NAME
  --name NAME_OF_KEY --value "YOUR_AZURE_AI_SERVICE_KEY"
```

Replace *KEY_VAULT_NAME* with the name of your key vault, *NAME_OF_KEY* with the key name you want to assign in the vault, and *YOUR_AZURE_AI_SERVICE_KEY* with the AI service key.

If you run into issues storing the Azure AI service key in the key vault, try adding your user account to the access policies in the Azure portal for your key vault and granting it full permissions.

Coding the system

Now that you have the key vault set up, you need can begin coding the system to implement the text analytics workload. Here, you'll define the entity recognition task and execute it to ensure the output meets your expectations.

Follow these steps:

1. Install the required Python packages by running the following command:

```
pip install azure-ai-textanalytics azure-identity azure-keyvault-secrets
```

2. Create a new file called *aisecurity2.py* and open it in your text editor. First, import the relevant libraries:

```
from azure.ai.textanalytics import TextAnalyticsClient
from azure.core.credentials import AzureKeyCredential
from azure.identity import DefaultAzureCredential
from azure.keyvault.secrets import SecretClient
```

3. Next, initialize your Azure credential with the following line:

```
credential = DefaultAzureCredential()
```

4. Use the following code to connect to Azure Key Vault and retrieve the Azure Text Analytics API key:

```
key_vault_name = "ai102practicalkeyvault"
key_vault_uri = f"https://{key_vault_name}.vault.azure.net/"
secret_client = SecretClient(vault_url=key_vault_uri, credential=credential)
secret_name = "AIKey"
ta_key = secret_client.get_secret(secret_name).value
```

Replace *key_vault_name* with the name of your key vault and the *secret_name* with your actual secret name.

5. Now, initialize the Text Analytics client:

```
ta_endpoint = "https://ai102sentimentendpoint.cognitiveservices.azure.com/"
text_analytics_client = TextAnalyticsClient(
    endpoint=ta_endpoint,
    credential=AzureKeyCredential(ta_key)
)
```

Replace *https://ai102sentimentendpoint.cognitiveservices.azure.com/* with your own endpoint value.

6. Add the following line so you have a document on which to run the entity recognition task:

```
documents = [
    "Microsoft was founded by Bill Gates and Paul Allen on April 4, 1975, "
    "to develop and sell BASIC interpreters for the Altair 8800."
]
```

7. Then add this line to perform the task:

```
response = text_analytics_client.recognize_entities(documents=documents)
```

8. Finally, add this code to print out the entities that are recognized:

```
for doc in response:
    print(f"Entities in document {doc.id}:")
    for entity in doc.entities:
        print(
            f"...Entity: {entity.text}, Category: {entity.category},\n"
            f"    Subcategory: {entity.subcategory}\n"
            f"    Confidence Score: {entity.confidence_score}"
        )
```

9. Run the script with:

```
python aisecurity2.py
```

You should see something like what's shown in Figure 2-6.

```
C:\Users\renal\Documents\Github\AI-102\chapter2>python aisecurity2.py
Entities in document 0:
...Entity: Microsoft, Category: Organization, Subcategory: None, Confidence Score: 0.99
...Entity: Bill Gates, Category: Person, Subcategory: None, Confidence Score: 1.0
...Entity: Paul Allen, Category: Person, Subcategory: None, Confidence Score: 1.0
...Entity: April 4, 1975, Category: DateTime, Subcategory: Date, Confidence Score: 1.0
...Entity: Altair 8800, Category: Product, Subcategory: None, Confidence Score: 0.98

C:\Users\renal\Documents\Github\AI-102\chapter2>
```

Figure 2-6. Output from running the entity recognition program

You'll see the different entities that were detected, along with their category, subcategory, and the confidence score provided by the prediction. In this case the confidence scores the entity recognition program returned are relatively high, indicating strong certainty that those are the appropriate entities. When entities are less clear, lower confidence scores will likely be returned.

With the output returning as expected, you can move on to the next step.

Monitoring and logging your solution

Remember, two key phases of the lifecycle of an AI project are maintenance and monitoring. Next, you'll set up the necessary monitoring to ensure your solution continues to perform as expected. This will provide important feedback, help you identify future improvements or optimizations, and assist withdebugging if issues arise while running the system.

Here are the steps to follow to set up monitoring:

1. Navigate to the Language resource that you have provisioned in the Azure portal.
2. In the left pane, under Monitoring, select Logs, then click the logs for your resource to open them.
3. Click Create to create a new Log Analytics workspace.
4. An overview of the workspace should open. You can set alerts under "Manage alert rules."
5. Click Create to create a new alert.
6. You will be prompted to select a signal. Enter "% Used Memory" as the signal name.
7. The alert logic settings should now appear. Select "static" for the threshold, "average" for the aggregation type, "greater than" for the operator, "count" for the unit, and "80" for the threshold value. Leave the other settings as they are.
8. Navigate to "Review + create," review the settings, and create the alert.
9. Navigate to Azure Monitor and view the alert there.
10. Try modifying the time ranges to see what your alert will look like over various possible time periods.
11. Experiment with other alerts, and think about which would work best for your AI system. Refer to the list of common metrics in the previous section, and try applying some of them to the monitoring that you are performing. Take note of which ones are useful in your use case.

When calling Azure AI services, you may encounter HTTP status code 429 (rate limit exceeded) or 401 (authentication failure). To handle rate limits, implement retry logic with exponential backoff: read the `Retry-After` header and make the system wait for the server-suggested amount of time before retrying, cap the maximum delay to prevent excessively long waits, add some randomized jitter to avoid synchronized retries, and limit the total retry count to three to five attempts. It's a good idea to log each retry attempt for visibility. You can also leverage built-in SDK retry policies where available and batch your requests or downsample large payloads to reduce call volume. For authentication errors, catch 401 responses, refresh tokens automatically using managed identities or the refresh token flow, and fall back to alternative

credentials stored securely in Azure Key Vault. You should also audit all authentication failures in your logging system and ensure that your key vault policies allow seamless secret rotation without code changes, to maintain service availability.

For a more production-ready approach, consider incorporating validation tests (e.g., data quality checks, performance tests under load), security validation (e.g., verifying that unauthorized requests are denied), and operational readiness (e.g., runbooks detailing steps to recover from failures). You can also simulate spikes in data volume to measure the system's response time and identify possible bottlenecks. If compliance is a concern, you can perform compliance scans (HIPAA, GDPR, etc.) with your own logs and data retention policies to ensure that your solution meets regulatory standards before it goes live.

And with that, you've integrated security mechanisms into your Azure AI solution! As you proceed with the implementation of different AI systems, keep in mind that security will always be an important component. As solutions grow more complex, the potential for vulnerabilities increases, requiring additional layers of control. This exercise has introduced you to the fundamentals, and it will provide a foundation you can build on as you expand your understanding of security. This chapter has only scratched the surface; there are many more controls to address, and we'll explore them gradually throughout this guide.

Also bear in mind that enterprise AI solutions often require robust disaster recovery (DR) strategies, which may involve replicating AI services or using multiregion deployments in active-active or active-passive configurations. Compliance requirements (e.g., SOC, HIPAA, GDPR) can further dictate where and how data must be stored and processed. You'll also need to monitor service-level agreements (SLAs) to meet your business commitments. While Azure services typically provide SLAs for uptime, you'll need to account for dependent services and architectural redundancies. Finally, architecture diagrams and decision frameworks (such as the Azure Well-Architected Framework) can help you and your team evaluate trade-offs among cost, complexity, and resilience when you're selecting services for data ingestion, storage, model hosting, or integration.

Chapter Review

In this chapter, we explored the lifecycle of developing an AI system on Microsoft Azure, focusing on how to integrate security and monitoring into AI workloads. We also examined deployment considerations, including how to implement a structured CI/CD pipeline for your AI services. By now, you should be familiar with the key tools and practices required to successfully integrate, secure, and deploy AI solutions in Azure.

To be successful on the AI-102 exam, you'll need to be able to complete the following tasks related to the key points we covered in this chapter:

- Integrate Azure AI services into a CI/CD pipeline for deployment.
- Configure diagnostic logging for AI services.
- Implement network-level security, both at a general Azure level and for specific AI services.
- Set up monitoring and alerts for AI services and understand core metrics used.
- Work with REST APIs and understand how to construct and interpret HTTP requests and JSON responses.
- Manage costs effectively and use tools relevant to cost monitoring and optimization.
- Follow best practices for managing account keys and protecting them in Azure Key Vault.
- Configure private network communication between endpoints in your AI solution.

In the next chapter, we'll build on this foundation by exploring the storage solutions available for AI workloads, covering how to store training data, state data, and other types of data you'll use when working with different AI services.

Chapter Quiz

1. Your team is initiating the requirements definition and design phase of a new Azure AI project that's intended to develop a predictive maintenance system for industrial equipment. What must the team do first to ensure the project's success?
   A. Define clear project objectives and success metrics.
   B. Design the user interface for system administrators.
   C. Develop an initial list of potential AI models to use.
   D. Plan the integration with existing IT infrastructure.
2. In the context of Azure AI projects, which project lifecycle phase involves evaluating the performance of the AI model and iterating to improve it?
   A. Requirements definition and design
   B. Monitoring
   C. Development
   D. Deployment