Student Helper: A RAG Application on Secure, Cost-Optimized AWS Infrastructure

An overview of the Infrastructure as Code (IaC) architecture designed for onboarding and knowledge sharing.

What it is

The complete cloud infrastructure for the “Student Helper” Retrieval-Augmented Generation application.

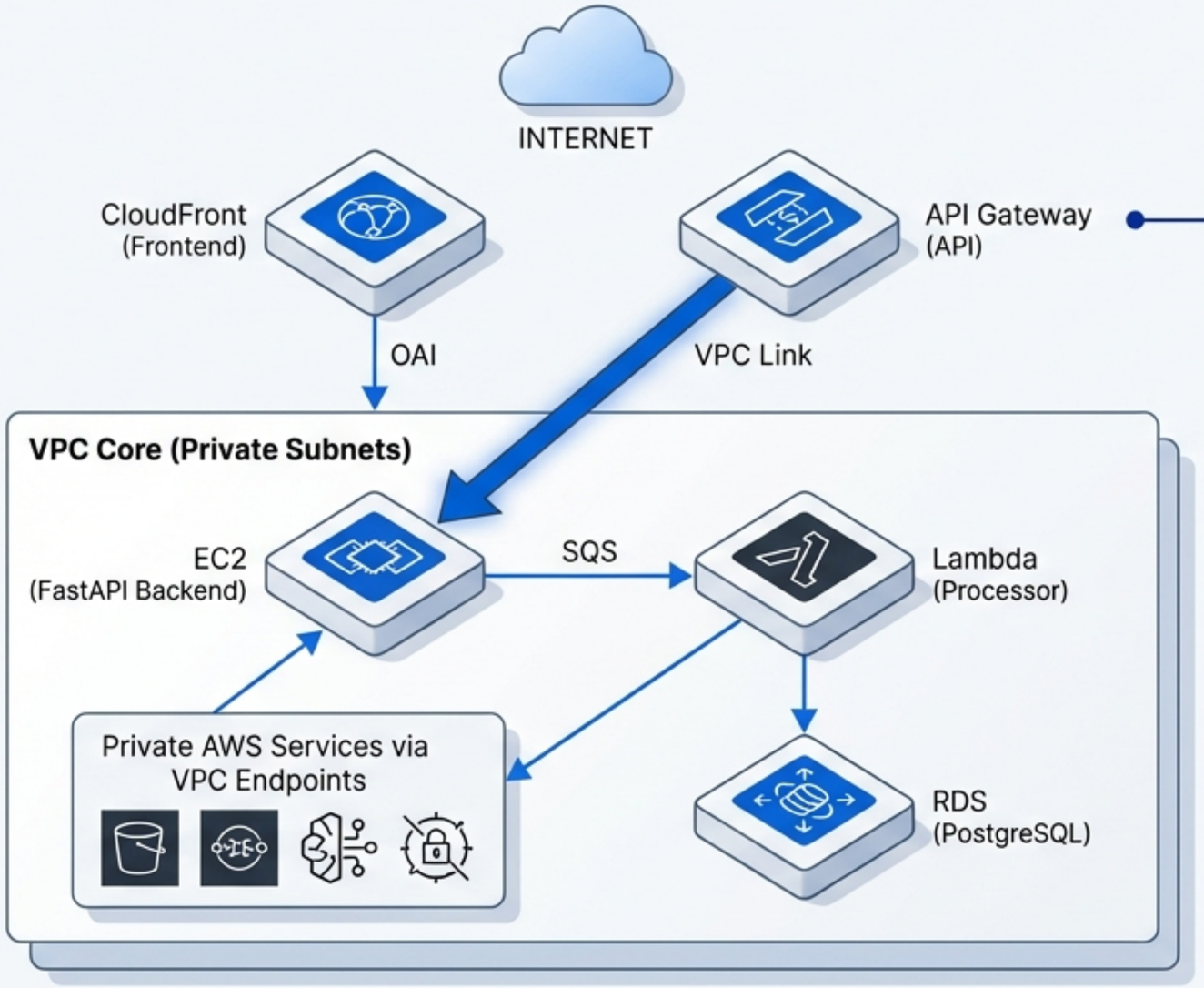
How it’s built

Implemented entirely with **Pulumi** using **Python**, enabling type-safe, component-based infrastructure management.

Core Philosophy

A private-first, security-conscious design that aggressively minimizes costs by avoiding common defaults like NAT Gateways.

The Complete System Architecture at a Glance

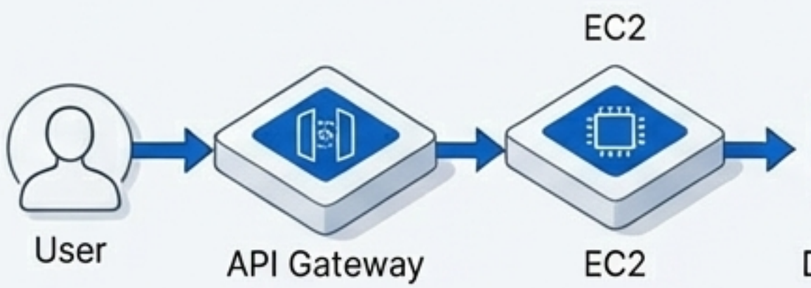


The system is divided into three logical zones:

the public-facing Edge, the secure VPC Core where all application logic resides, and the private connections to other AWS services.

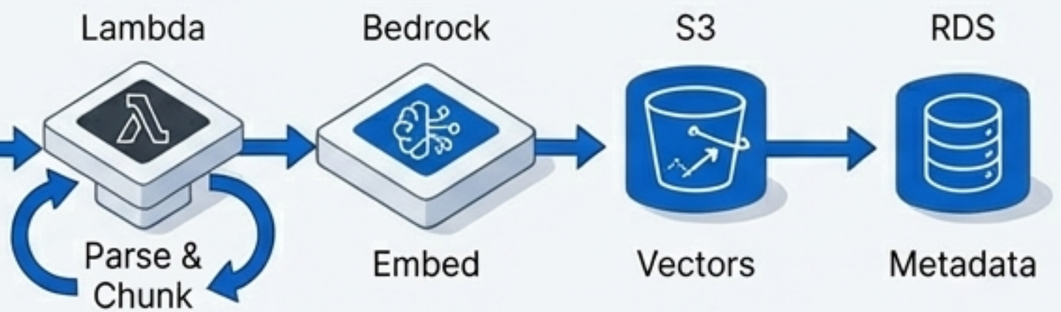
Following the Data: From User Upload to AI-Generated Answer

1. Document Ingestion



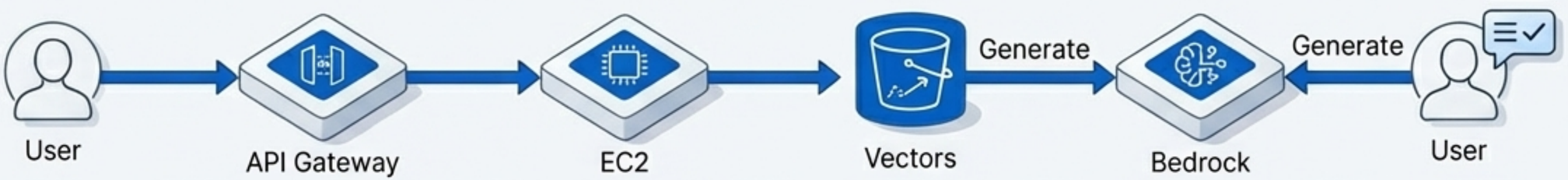
The API receives the file and triggers an asynchronous processing job.

2. Asynchronous Vectorization



A decoupled Lambda function handles the heavy lifting of document parsing and embedding, without blocking the API.

3. Question Answering



The backend queries the vector store for relevant context and uses a second LLM to synthesize the final answer.

Our Guiding Principles: Three Core Decisions Shaping the Infrastructure

No NAT Gateway

Decision: All outbound AWS service access is handled by VPC Endpoints.

Rationale: Significant cost savings (~75% reduction in gateway fees) and improved security by keeping all traffic on the AWS private backbone.

Private API via VPC Link

Decision: The EC2 backend has no public IP address; API Gateway connects to it privately.

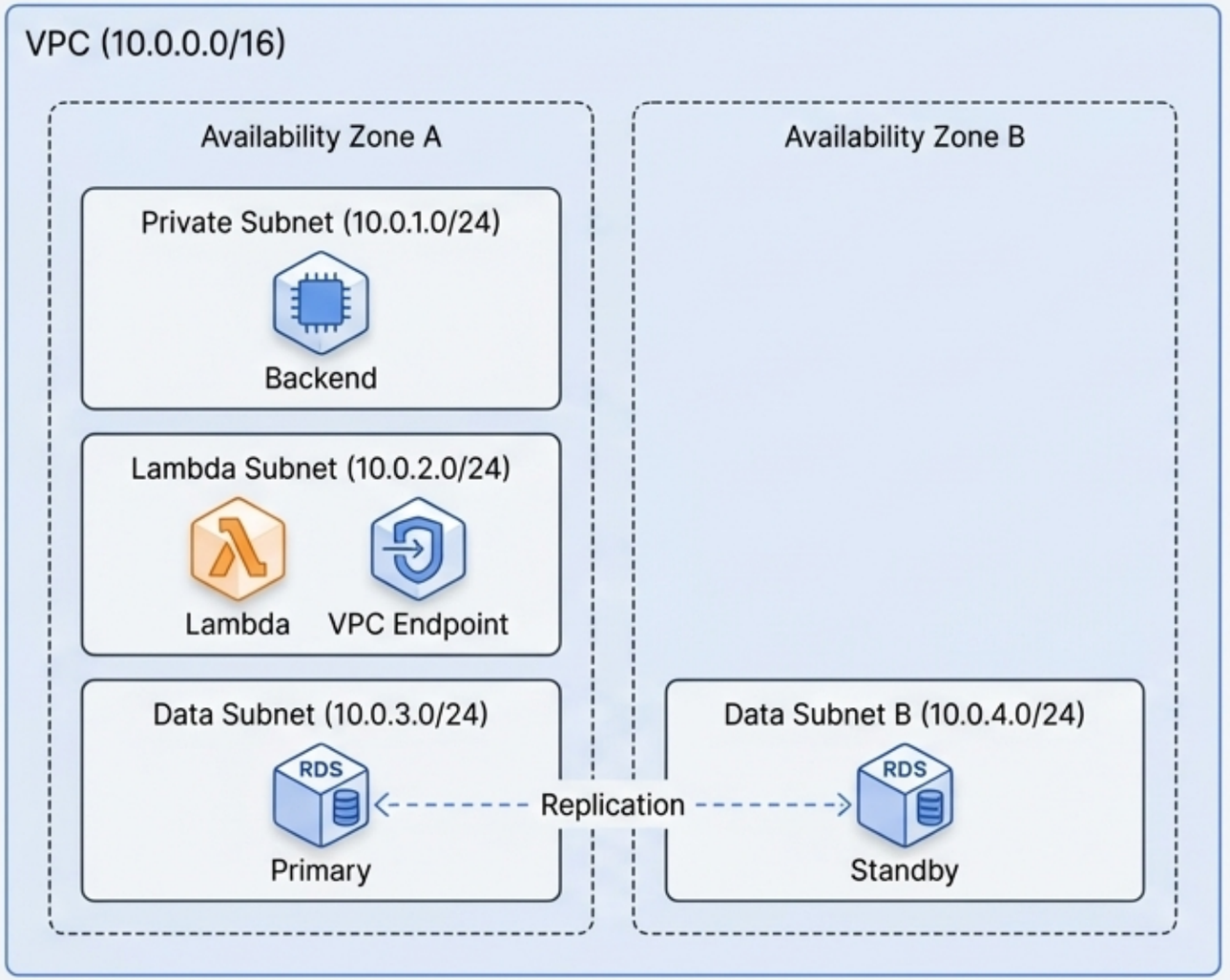
Rationale: Reduces the instance's attack surface. All traffic is funneled through the managed, secure API Gateway service.

Native Vector Storage with S3 Vectors

Decision: Use the new S3 Vectors feature instead of a dedicated vector database like Pinecone or Weaviate.

Rationale: Simplifies the architecture (no separate service to manage), reduces operational overhead, and provides a cost-effective solution integrated directly with S3.

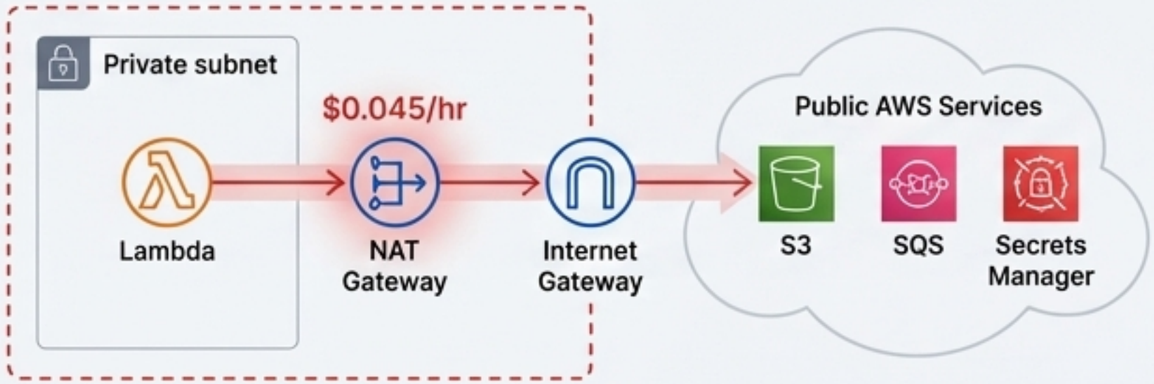
The Network Foundation: A Private-First VPC



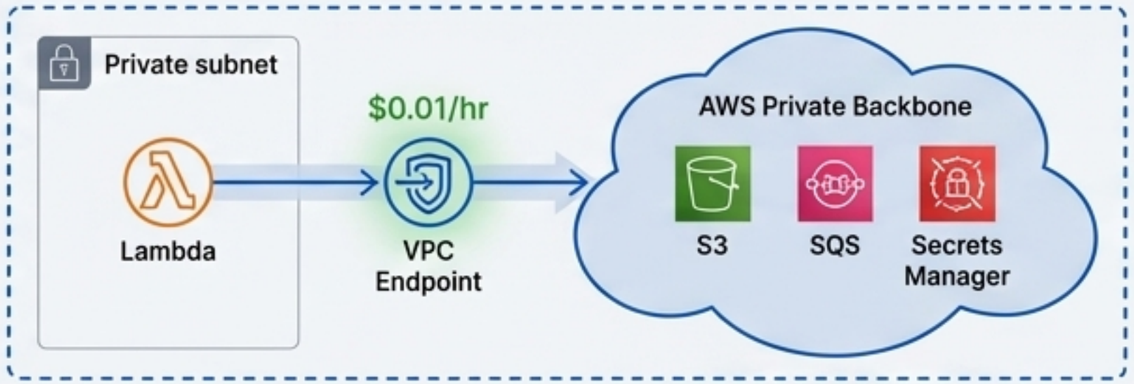
- **Isolation**: The entire application runs within a logically isolated Virtual Private Cloud.
- **No Public Subnets**: Compute and data resources are not directly accessible from the internet.
- **Purpose-Driven Subnets**: Subnets are segmented by function (compute, lambda, data) for better organization and routing control.
- **High Availability**: The data subnets span two Availability Zones, a requirement for for Multi-AZ RDS deployments.

How VPC Endpoints Provide Secure, Low-Cost AWS Service Access

Traditional Architecture



Our Architecture



Endpoint Types Explained

Gateway Endpoints (Free)

For S3 & DynamoDB. Works by adding a prefix list entry to the subnet's route table.

Route Table

Destination	Target
10.0.0.0/16	local
pl-xxx (S3)	vpce-xxx

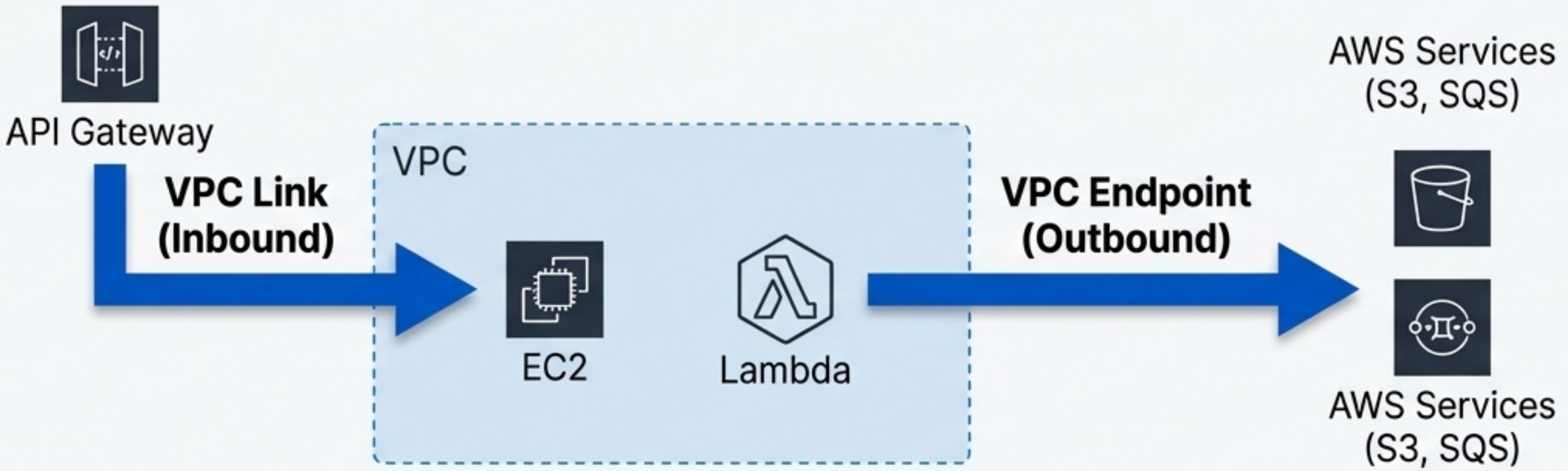
Interface Endpoints (Paid)

For most other services (SQS, Bedrock, Secrets Manager). Works by placing an Elastic DNSI and using Private DNS.

Benefits

- **Cost:** ~75% reduction on gateway charges.
- **Security:** Traffic never traverses the public internet.
- **Performance:** Lower latency by staying on the AWS network.

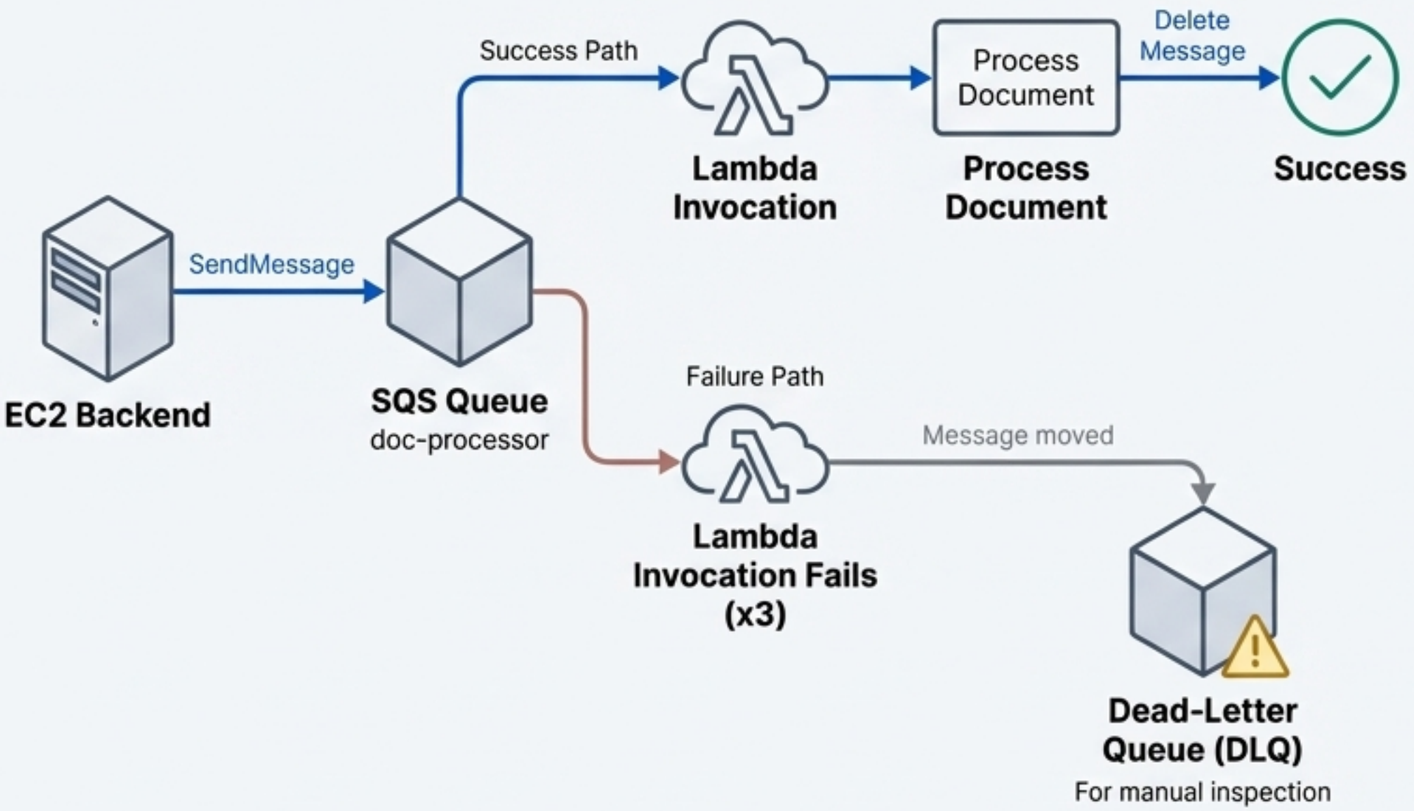
Inbound vs. Outbound: VPC Link and VPC Endpoints



Feature	VPC Endpoint	VPC Link
Direction	**Outbound** (VPC → AWS Services)	**Inbound** (API GW → VPC)
Use Case	Lambda function calling S3 or SQS.	External user calling your private EC2 API.
Mechanism	Creates a route entry or an ENI in your subnet.	Creates an ENI for API Gateway to use.

Endpoints let your code call AWS. **Links** let AWS call your code.
Both are essential for a fully private architecture.

The Asynchronous Engine: Decoupling Ingestion with SQS and Lambda



Key Configuration Details

- **Visibility Timeout:** `360 seconds`. Set to be longer than the Lambda timeout to prevent duplicate processing.
- **Max Receive Count:** `3` . A message is tried up to 3 times before being sent to the DLQ.
- **Batch Size:** `1`. The Lambda function is triggered with a single document at a time for simpler error handling.

```
# Pulumi: Configuring the Redrive Policy
self.queue = aws.sqs.Queue(
    "doc-processor",
    ...
    redrive_policy=self.dlq.arn.apply(
        lambda arn: json.dumps({
            "deadLetterTargetArn": arn,
            "maxReceiveCount": 3,
        })
    ),
)
```

The Data Layer: RDS for Metadata, S3 Vectors for Search

RDS PostgreSQL for Metadata

Stores document metadata, user sessions, and processing status.

Key Features

- **Engine**: PostgreSQL 16
- **HA**: `multi_az=True` for production stacks.
- **Security**: `storage_encrypted=True`, accessed only via a dedicated security group.
- **Credentials**: Master password is automatically managed and rotated by AWS Secrets Manager.

S3 Vectors for Embeddings

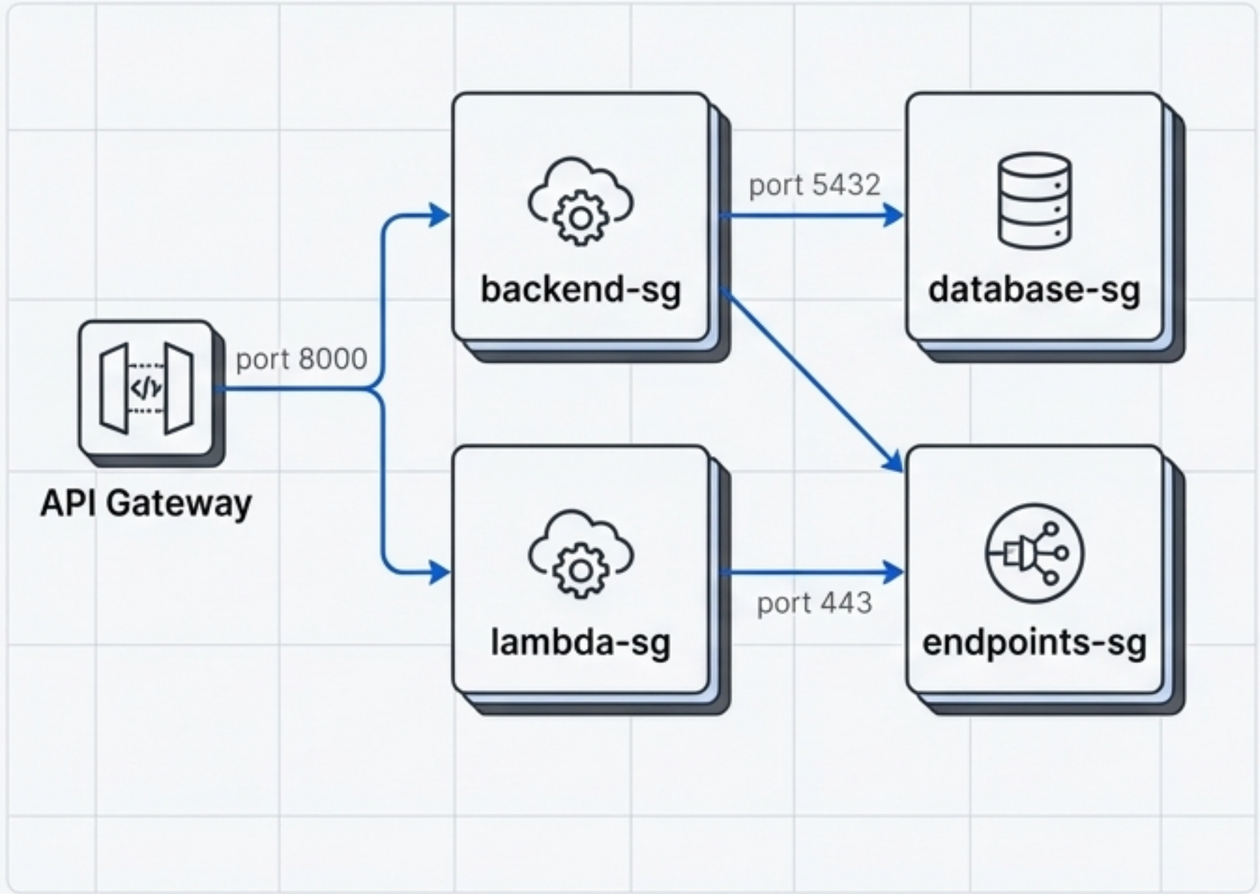
Stores document embeddings for efficient semantic search.

```
# S3 Vector Index
self.vectors_index = aws_native.s3vectors.Index(
    ...
    dimension=1536, # Titan v2
    distance_metric="COSINE",
    metadata=[
        {"name": "document_id", "data_type": "STRING", "filterable": True},
        {"name": "page_number", "data_type": "NUMBER", "filterable": True},
    ],
)
```

This native AWS solution avoids the complexity and cost of a separate vector database service, simplifying the overall architecture.

A Multi-Layered Security Strategy

Security Group



Defense in Depth Layers

Layer 1: Edge

- CloudFront (DDoS Protection, HTTPS), API Gateway (Rate Limiting, CORS).

Layer 2: Network

- Isolated VPC, Private Subnets, Security Groups (Least-Privilege), VPC Endpoints (No Internet Exposure).

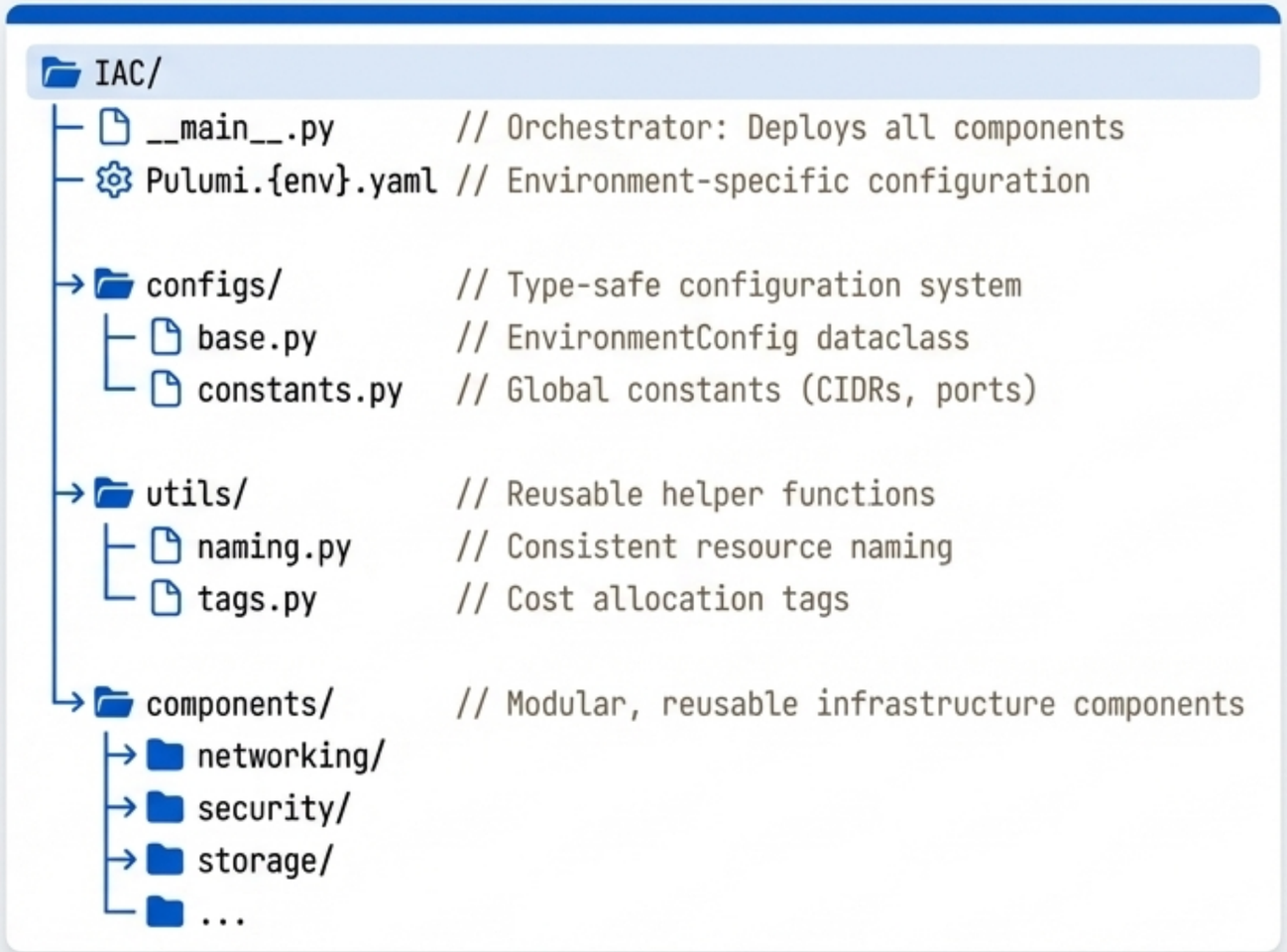
Layer 3: Identity

- IAM Roles (Least-Privilege), EC2 Instance Profiles (No Hardcoded Credentials).

Layer 4: Data

- Encryption at Rest (S3, RDS) and in Transit (TLS), IMDSv2 enforced on EC2 to prevent SSRF.

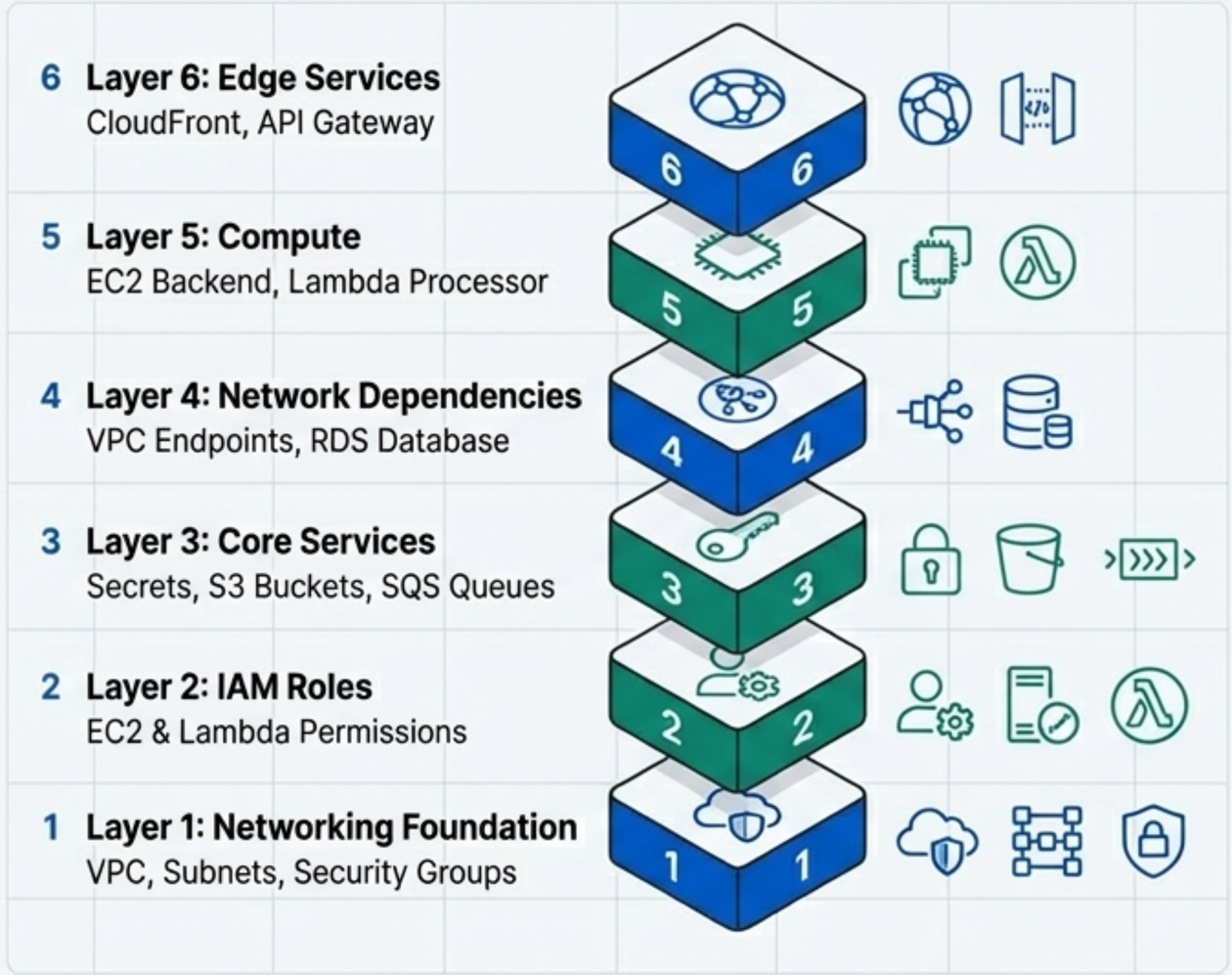
The Code Behind the Cloud: A Structured Pulumi Project



Key Principles

- **Orchestration**: `__main__.py` defines the deployment sequence, acting as the entry point.
- **Modularity**: Resources are grouped into logical `ComponentResource` classes inside the `components/` directory.
- **Configuration**: All environment-specific values (instance sizes, domain names) are externalized into stack configuration files and loaded via a type-safe dataclass.

Building with Components in a Layered Deployment



__main__.py

```
# Layer 1: Networking Foundation
vpc = VpcComponent(...)
security_groups = SecurityGroupsComponent(...)

# Layer 2: IAM Roles
iam_roles = IamRolesComponent(...)

# ... and so on for each layer, showing
# the sequential instantiation of components.
```

This layered approach ensures resources are created in the correct order, managing dependencies implicitly and making the deployment process predictable and reliable.

Configuration-Driven Infrastructure for Multiple Environments

Type-Safe Configuration

We use a Python dataclass (EnvironmentConfig) to load and validate all settings from Pulumi's stack configuration. This prevents typos and ensures all required values are present.

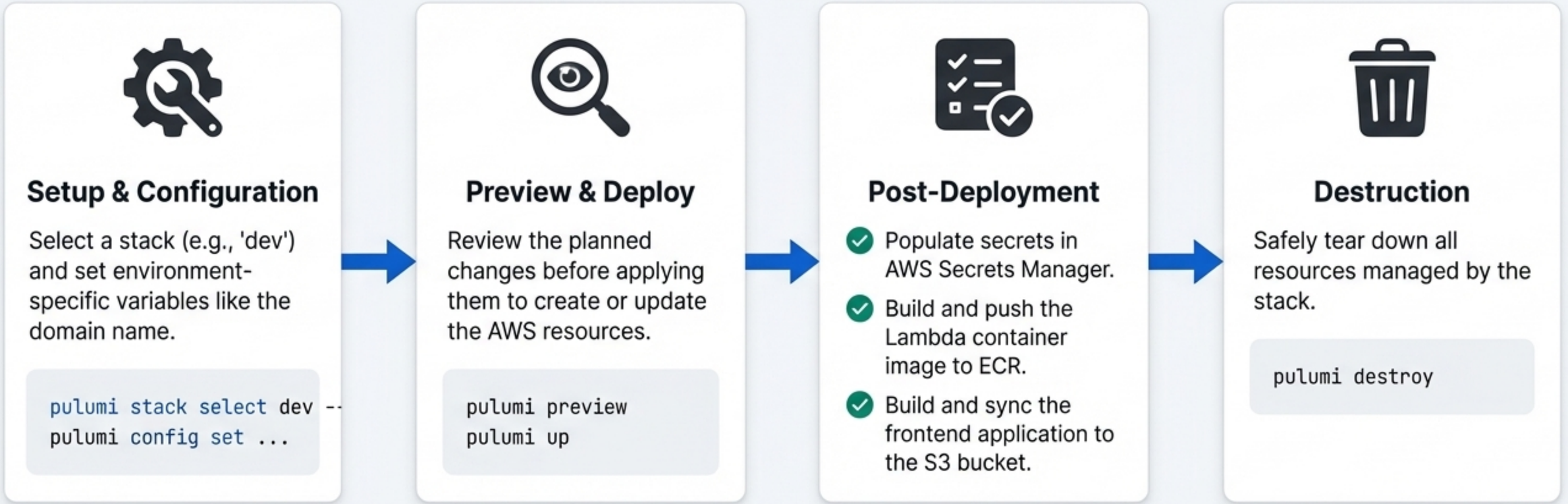
```python
@dataclass
class EnvironmentConfig:
    environment: str
    domain: str
    ec2_instance_type: str
    rds_instance_class: str
    multi_az: bool
    # ... and other key fields
```

Environment Matrix

A practical example of how the configuration drives differences between environments.

Setting	Dev	Prod
EC2 Instance	`t3.micro`	`t3.small`
RDS Instance	`db.t3.micro`	`db.t3.medium`
RDS Multi-AZ	No	**Yes**
Deletion Protection	No	**Yes**
Backup Retention	1 day	**7 days**

From Code to Cloud: The Deployment Workflow



Architecture in Review: Key Decisions and Their Impact

Decision	Our Choice	Rationale & Impact
IaC Tool	**Pulumi with Python**	Enabled a component-based, type-safe model Enabled a component-based, type-safe model that aligns with our application language and improves maintainability.
API Backend	**EC2 + Lambda**	Chose the right tool for the job: always-on EC2 for the interactive API, and event-driven Lambda for scalable, asynchronous processing.
Vector DB	**S3 Vectors**	Leveraged a native AWS service to reduce architectural complexity and operational cost, avoiding a separate database.
Private Networking	**VPC Endpoints**	Eliminated the need for a costly NAT Gateway, resulting in a more secure, lower-latency, and cost-effective design.