CIH Solution

CIH Solution: Architecture & Build Process

Technical Documentation for IT Professionals

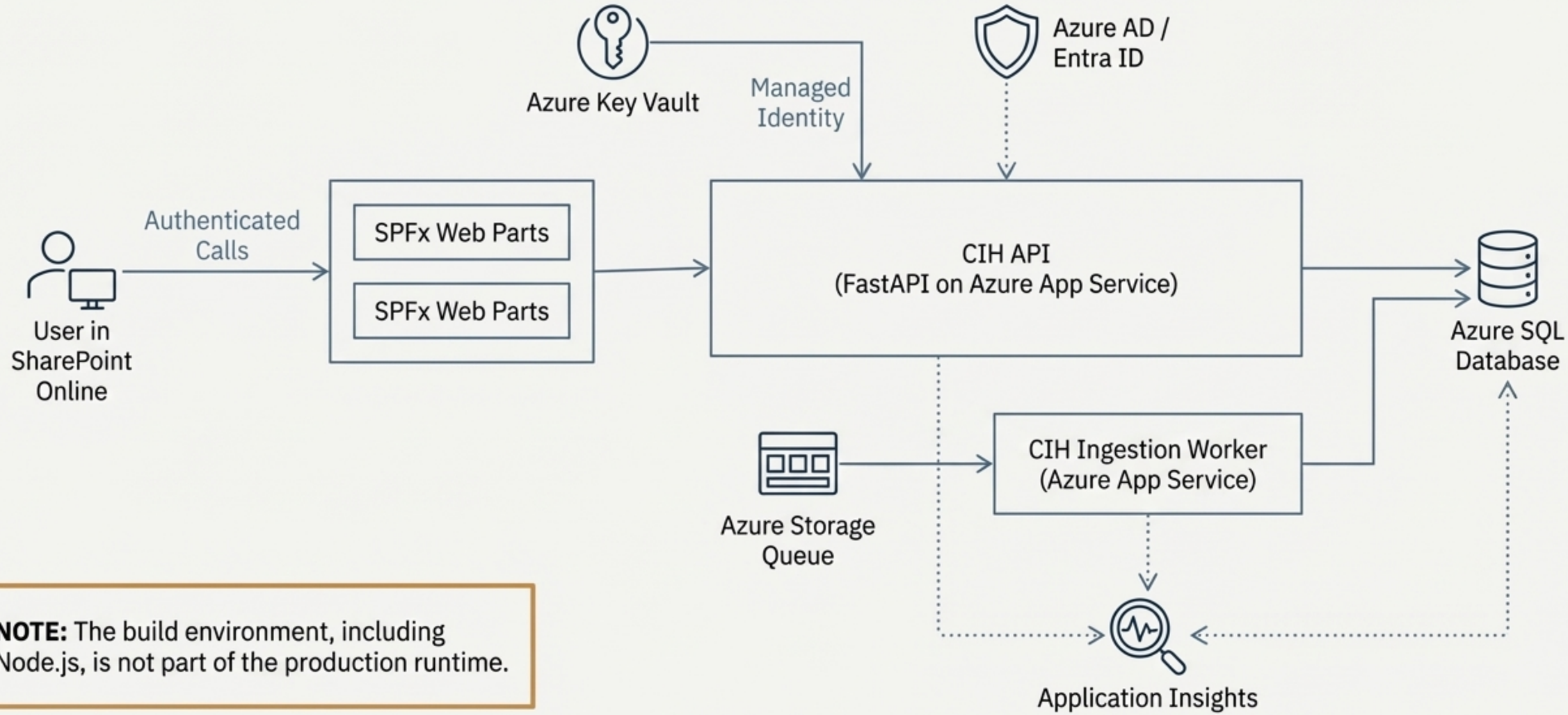
A Clear Separation Between Build and Production

Node.js and its related tooling are used exclusively during the development and packaging of the SharePoint Framework (SPFx) user interface. They ensure the application is built using a supported, stable toolchain.

Once the SharePoint package is created and deployed, neither Node.js nor any build-time tools are required in the customer’s production environment.

This approach aligns with enterprise deployment standards, reduces the operational footprint, and minimizes security risk.

CIH Production Runtime Architecture



NOTE: The build environment, including Node.js, is not part of the production runtime.

Solution Component Inventory

SharePoint Framework UI (SPFx)

Purpose: User-facing web parts running within the customer's SharePoint Online environment.

Components: Dashboard Web Part, Employee Search Web Part.

Technology: Built with TypeScript/React, bundled into a SharePoint package (.sppkg).

CIH Backend API

Purpose: Secure API layer that validates tokens, enforces RBAC, and provides data to the UI.

Technology: FastAPI running on an Azure App Service.

Azure Platform Services

Purpose: Provides the secure, scalable, and resilient infrastructure foundation.

Services:
Azure Key Vault (for secrets management)
Azure SQL / Storage (for data persistence and job queuing)
Application Insights (for monitoring and diagnostics)

The Distinction: Build-Time vs. Runtime Environments

Build-Time Environment (Developer / CI Machine)

To compile, bundle, and package the SPFx application.

Tools Used:

- ✓ Node.js
- ✓ npm (Node Package Manager)
- ✓ Yeoman (SPFx generator)
- ✓ Gulp (Task runner)
- ✓ TypeScript / React Toolchain

Output: A deployable .sppkg package file.

Production Runtime Environment (Customer Azure/M356 Tenant)

To host and run the live application.
IBM Plex Sans Regular.

Components:

- ✓ SharePoint Online
- ✓ Azure App Service (hosting the FastAPI)
- ✓ Azure AD / Entra ID

Key Fact:

- ✗ Node.js is NOT required.
- ✗ npm is NOT required.
- ✗ nvm is NOT required.

The Role of Node.js: The Build Engine

Think of Node.js as the factory that assembles the product. It is essential for manufacturing, but it is not part of the finished product itself.

Key Responsibilities

- **Project Scaffolding:** Uses the Yeoman generator (`yo @microsoft/sharepoint`) to create the standard SPFx project structure.
- **Dependency Management:** Uses `npm` to install the required libraries and frameworks.
- **Code Compilation:** Compiles modern TypeScript and React code into browser-compatible JavaScript.
- **Asset Bundling:** Bundles all JavaScript, CSS, and other assets into optimized files for efficient loading in SharePoint.
- **Package Creation:** Produces the final, deployable SharePoint package (`.sppkg` file).

Ensuring Build Consistency with NVM (Node Version Manager)

The SharePoint Framework (SPFx) build tools have a strict dependency on specific Node.js versions. Using an incorrect version will break the build process.

SPFx & Node.js Compatibility Table

Node Version	SPFx Status
16.x	⚠️ Legacy
18.x (LTS)	✅ **Supported / Recommended**
20.x	❌ Breaks SPFx Build
22+	❌ Unsupported

The Solution: NVM as a Version Safety Switch

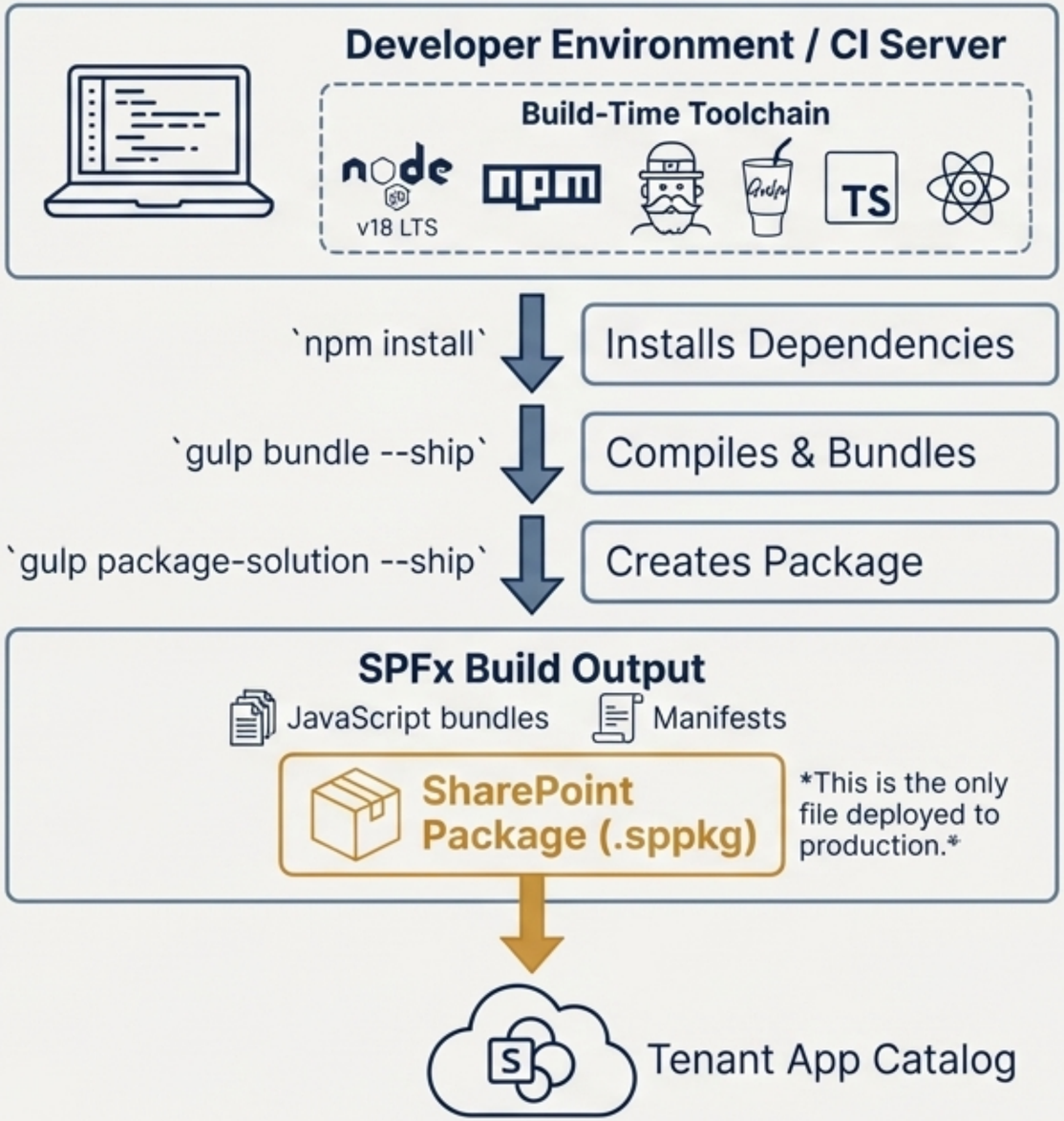
What it does
`nvm` allows developers to install and switch between multiple Node.js versions on a single machine.

Why it matters
It prevents accidental global upgrades of Node.js from breaking the project build. It ensures every developer and the CI/CD pipeline uses the exact same, validated version, guaranteeing consistent and repeatable builds.

Example Command

```
nvm use 18.20.8
```

The SPFx Build Process Flow



End-to-End Deployment Sequence

The solution is deployed in a specific order to ensure dependencies are met and security is correctly configured.

1 Provision Azure Resources
Deploy the core infrastructure using the provided Bicep template.

2 Configure Secrets
Create the required secrets in the newly deployed Azure Key Vault.

3 Prepare Database
Deploy the Azure SQL DB and run the schema script (DDL).

4 Deploy Backend Services
Deploy the API and Worker applications to Azure App Service and configure their settings.

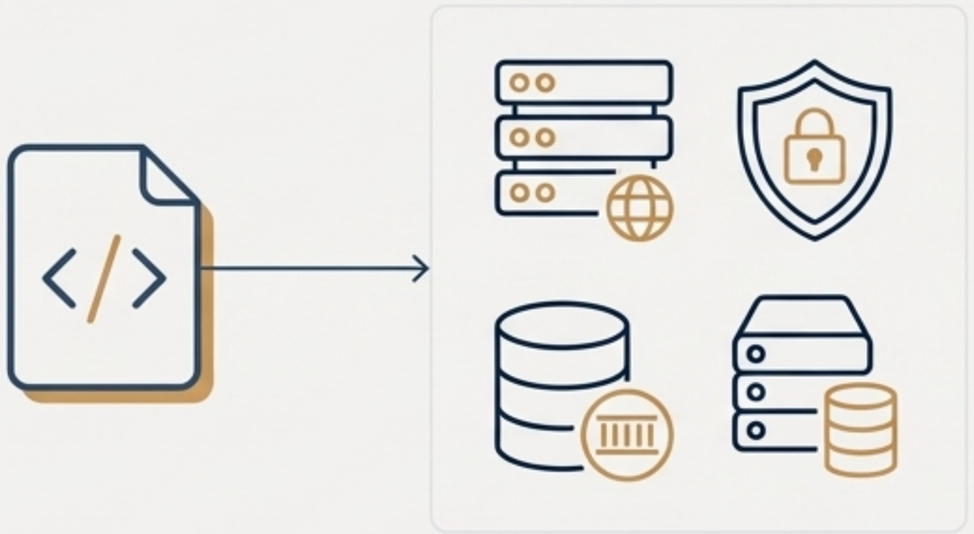
5 Configure Entra ID
Finalize the Entra ID app registration for API permissions.

6 Deploy SharePoint UI
Upload the `.sppkg` package to the SharePoint App Catalog and approve permissions.

7 Validate & Test
Run validation scripts and perform end-to-end testing.

Step 1: Provisioning Azure Infrastructure via Bicep

All Azure infrastructure is defined as code (IaC) to ensure consistent, repeatable, and automated deployments.



What's Deployed:

- App Service Plan + Web Apps (for API & Worker)
- Storage Account + Queue
- Key Vault (RBAC enabled)
- Application Insights + Log Analytics

Key Security Features Implemented by Bicep:

- **Managed Identity RBAC**: The API's App Service is granted **`Key Vault Secrets User`** role, eliminating the need for connection strings.
- **Key Vault References**: Application settings in the App Service securely reference secrets in Key Vault instead of storing them as plaintext values.

How to Implement:

1. Install Azure CLI (or use Azure Cloud Shell).
2. Edit `infra/bicep/main.parameters.json` to set environment-specific values (`prefix`, `resourceGroupName`).
3. Run the deployment command:

```
az deployment sub create \
  --location australiaeast \
  --template-file infra/bicep/main.bicep \
  --parameters @infra/bicep/main.parameters.json
```

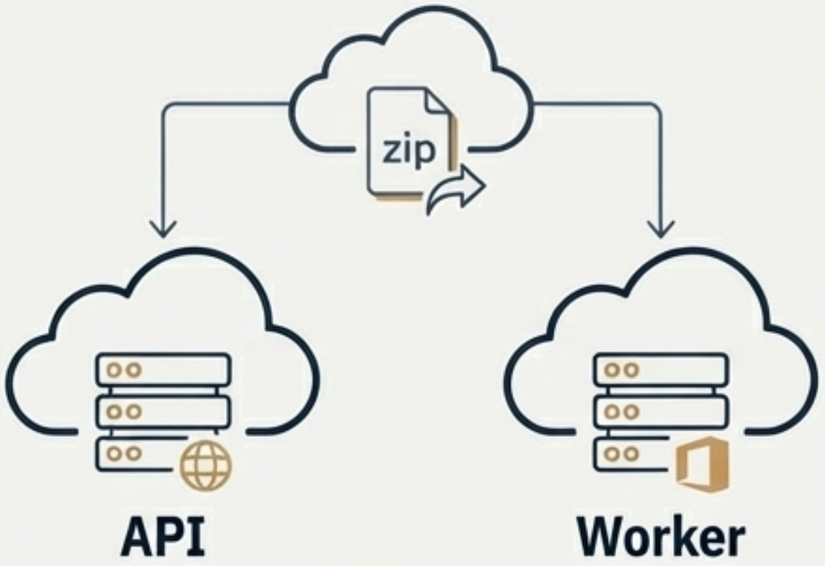
Step 2: Deploying the API and Worker

Deployed the API and secret irturuss and reacolovering the API and Worker for contal.

API & Worker Deployment

Method: Zip deploy is the recommended method for simplicity and reliability.

Hosting: The API and Worker are deployed as separate App Services for isolation and independent scaling.



Configuration & Validation

Critical Application Settings (in App Service → Configuration):
These settings link the application to the Azure AD tenant and define RBAC policies.

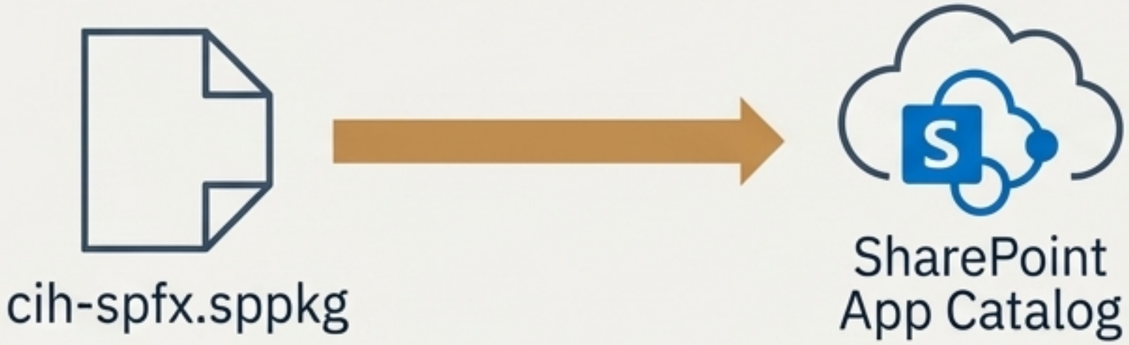
- `AAD_TENANT_ID`
- `AAD_API_AUDIENCE` (The Application ID URI of your API's app registration)

Optional RBAC enforcement:

- `REQUIRED_AAD_GROUP_IDS` (comma-separated)
- `REQUIRED_APP_ROLES` (comma-separated)

Validation:

- After deployment, navigate to **Diagnose and solve problems** in the App Service to confirm that Key Vault references are resolving successfully.
- Test the `/health` endpoint with a valid bearer token to confirm connectivity.



Step 3: Deploying the SharePoint (SPFx) Package

The `.sppkg` file, created during the build process, is deployed to the tenant-wide SharePoint App Catalog. This makes the web parts available for use on any site.

1. Navigate to the SharePoint Admin Center.
2. Open the App Catalog site.
3. Upload the `cih-spfx.sppkg` file from the `spfx/cih-spfx/sharepoint/solution` folder.
4. In the confirmation dialog, ensure the option ‘Make this solution available to all sites in the organization’ is checked.
5. Click **Deploy**.
6. **Approve API Permissions**: Go to the ‘API access’ page in the SharePoint Admin Center. Approve the pending permission request for the solution to call the CIH API. This is a one-time, tenant admin action.

Step 4: Final Configuration & Validation

1. Create Key Vault Secrets

After the Bicep deployment, manually create the required secrets in the new Key Vault. Secret names must match what the application expects:

- `concur-client-secret`
- `concur-refresh-token`
- `sql-password`

2. Configure SPFx Web Parts

Add the 'CIH Dashboard' and 'Employee Search' web parts to a SharePoint page. Edit each web part's properties and set:

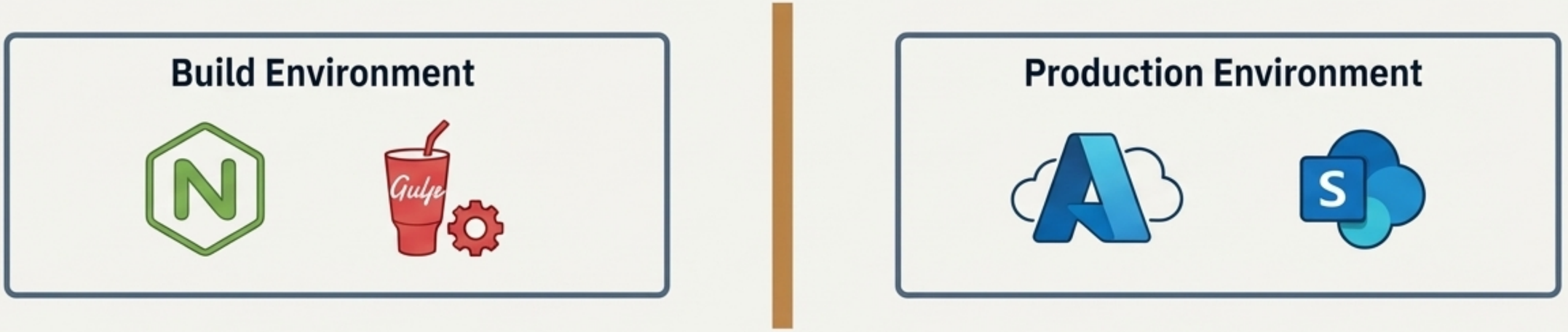
- **API Base URL**: `https://<your-api-app-name>.azurewebsites.net`
- **API Resource**: The Application ID URI (e.g., `api://<api-app-id>`)

3. Run Validation Scripts

The `CIH-Customer-Validation-Bundle.zip` contains PowerShell scripts to programmatically programmatically confirm:

- API is reachable and auth is configured.
- SQL database is reachable.
- Key Vault secrets exist.

A Secure, Decoupled, and Enterprise-Ready Architecture



- **Zero Production Footprint:** The Node.js toolchain never runs in the customer environment, reducing attack surface and operational overhead.
- **Repeatable & Auditable Infrastructure:** Infrastructure-as-Code (Bicep) ensures consistent deployments and simplifies change management.
- **Robust Security Posture:** Secrets are managed centrally in Key Vault with Managed Identity, eliminating credentials from code and configuration.
- **Clear Separation of Concerns:** The UI, API, and data layers are decoupled, allowing for independent maintenance and scaling.