Revolutionising DevOps with AI

From Pipelines to Deployment

10th May 2025

AJ Bajada

Azure, DevOps and automation enthusiast

And of course… Star Wars!

GitHub handle: tw3lveparsecs

Agenda

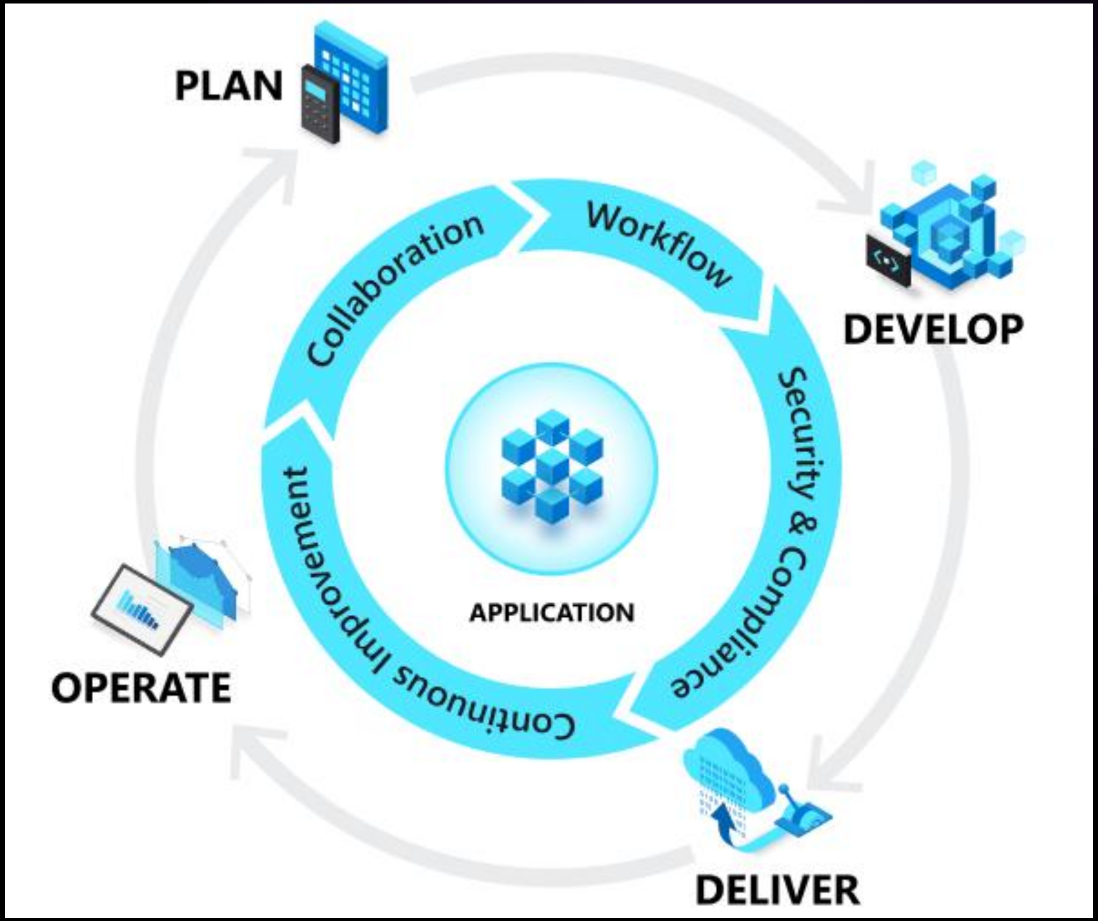
- Building AI-Powered DevOps Pipelines
- Lint Tests
- Deploying AI Infrastructure and Models
- PSRule for Azure

Agenda
AI
Building AI-Powered DevOps Pipelines
Lint Tests
Deploying AI Infrastructure and Models

Building AI-Powered DevOps Pipelines

What are DevOps Pipelines?

- DevOps pipelines are a series of automated processes that help development and operations teams build, test, and deploy software efficiently.
- They are designed to streamline the software development lifecycle, ensuring faster delivery and higher quality.



Challenges in Traditional DevOps Pipelines

- **Repetitive Manual Tasks**: Time-consuming activities like writing scripts and repeating boilerplate code.
- **Human Errors**: Misconfigurations and manual workflows increase risks of failures.
- **Inefficient Resource Utilisation**: Over-/under-provisioning wastes resources or impacts performance.
- **Testing Challenges**: Slow test creation, limited coverage, and delayed bug prioritisation affect quality.

AI the Game-Changer

- **October 2021:** GitHub Copilot was released for general use, delivering on-demand code suggestions and autocomplete features that not only complete individual code lines but also offer full code blocks tailored to your current context.
- **Copilot Chat:** This interactive conversational tool enables developers to directly engage with their code—allowing them to ask questions about the codebase, clarify specific functions, or request coding advice and guidance.

GitHub Copilot

Vibe Coding

- **Vibe coding** is an AI-dependent programming practice where a programmer describes a problem in a few sentences as a prompt
- This method allows software to be quickly created and debugged, enabling rapid prototyping without getting bogged down by the details of the generated code.

Agentic AI

Agentic AI exhibits autonomy, goal-driven behaviour, and adaptability.

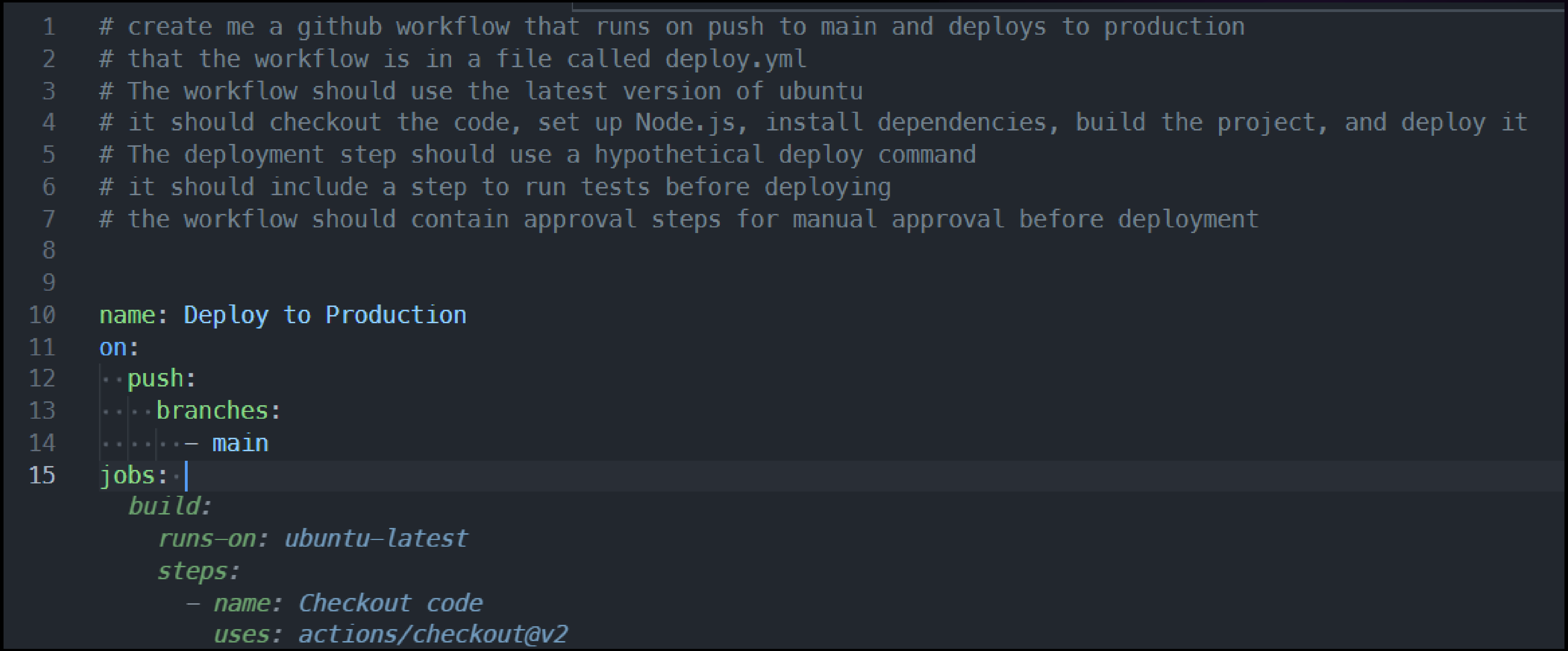
Where AI tools go beyond suggestions — they act as independent agents that can make decisions, take actions, and build software with minimal human prompts.

It writes code, tests it, handles errors, refactors, and even generates docs — all on its own.

AI Integration into DevOps Pipelines

Code Completion and Pipeline Script Generation	• AI tools like GitHub Copilot auto-generate YAML scripts and pipeline configurations, streamlining setup and reducing errors.
Automated Test Generation	• AI creates unit, integration, and end-to-end test cases by analysing code changes, prioritising critical areas to improve test coverage efficiently.
Intelligent Error Detection and Debugging	• AI analyses pipeline scripts and source code for errors or inefficiencies, offering fast debugging solutions through log analysis.
Monitoring and Continuous Improvement	• AI-driven analytics identify anomalies in performance metrics and provide actionable insights for refining pipeline configurations.

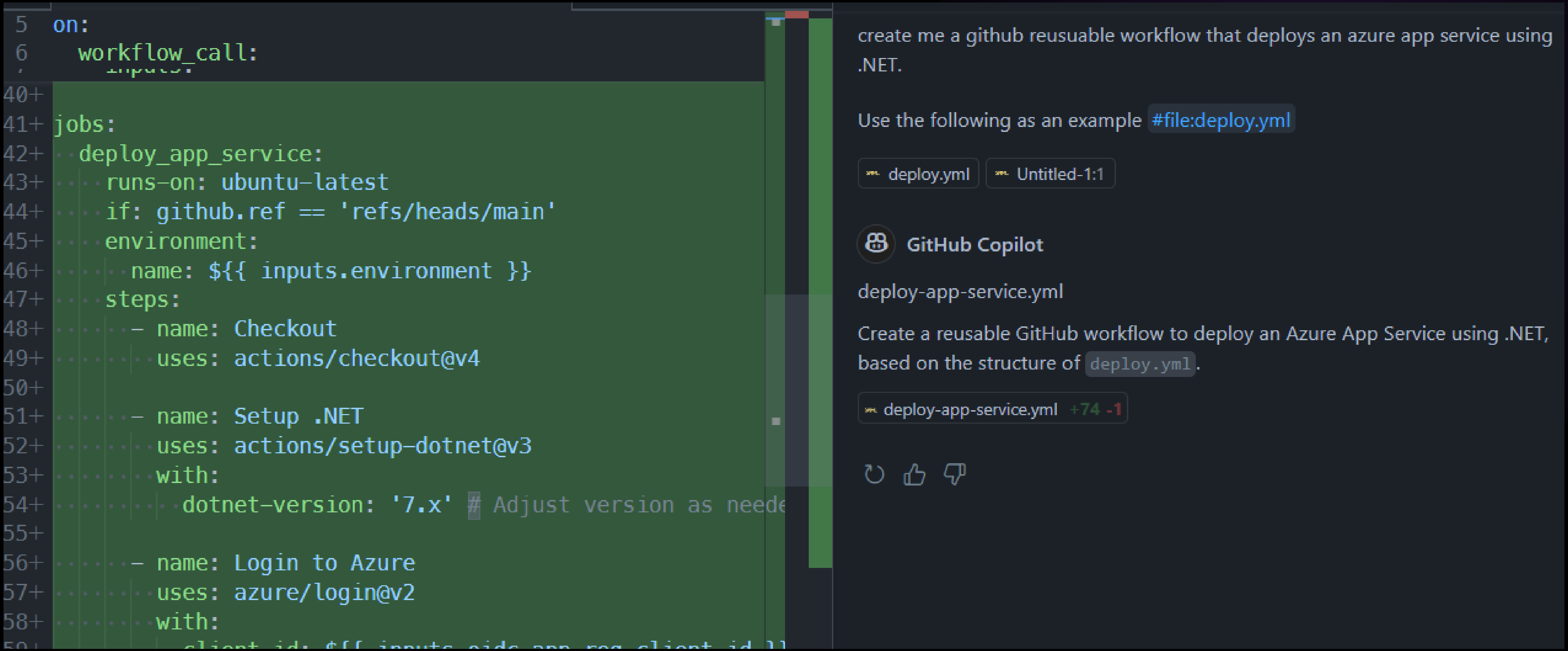
Building pipelines with code completion



```yaml
# create me a github workflow that runs on push to main and deploys to production
# that the workflow is in a file called deploy.yml
# The workflow should use the latest version of ubuntu
# it should checkout the code, set up Node.js, install dependencies, build the project, and deploy it
# The deployment step should use a hypothetical deploy command
# it should include a step to run tests before deploying
# the workflow should contain approval steps for manual approval before deployment


name: Deploy to Production
on:
  push:
    branches:
      - main
jobs: 
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
```

Building pipelines with chat



```
on:
  workflow_call:
    inputs:

jobs:
  deploy_app_service:
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment:
      name: ${{ inputs.environment }}
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup .NET
        uses: actions/setup-dotnet@v3
        with:
          dotnet-version: '7.x' # Adjust version as neede

      - name: Login to Azure
        uses: azure/login@v2
        with:
          client-id: ${{ inputs.aids-app-reg-client-id }}
```

create me a github reusable workflow that deploys an azure app service using .NET.

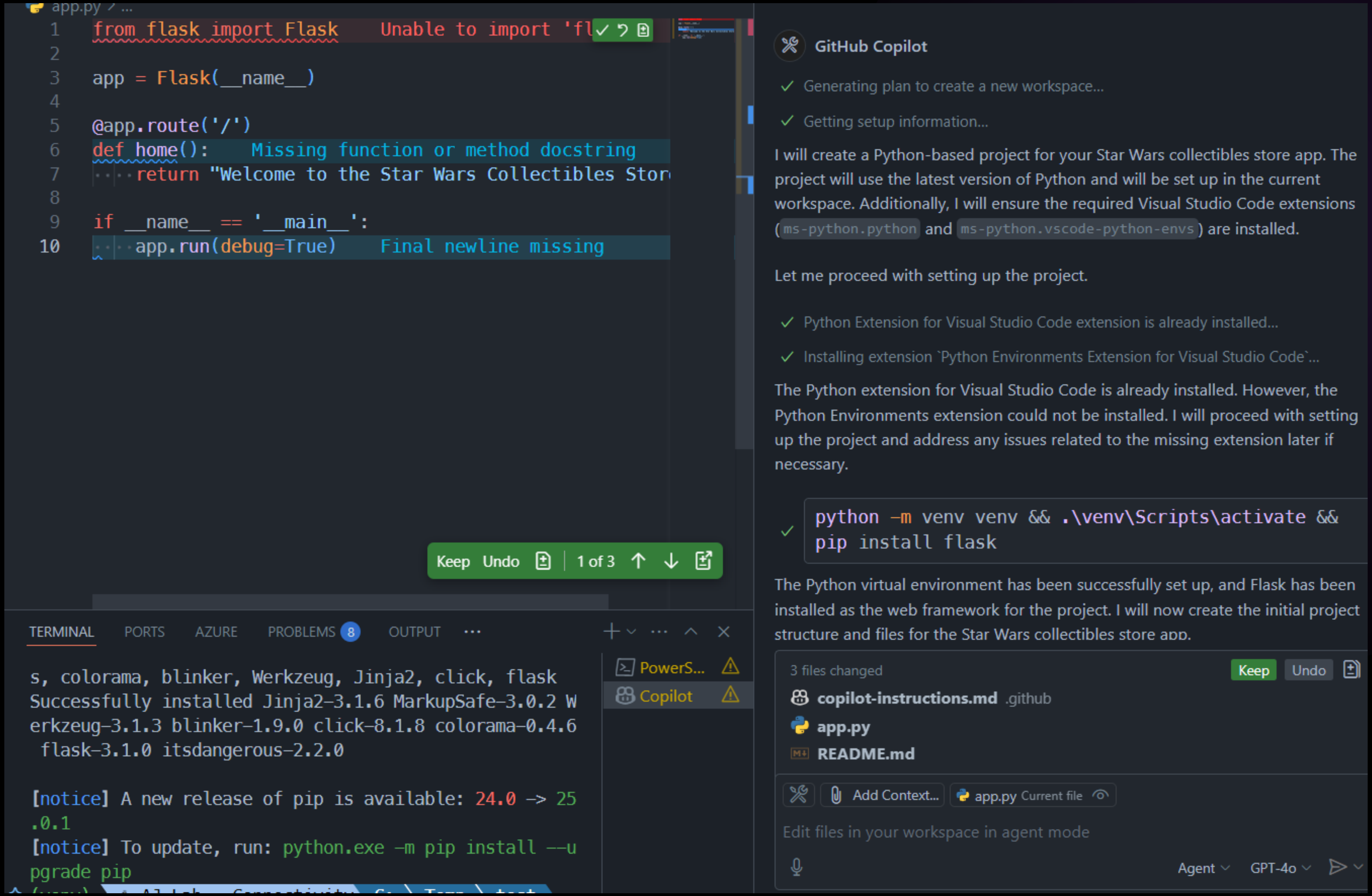
Use the following as an example #file:deploy.yml

GitHub Copilot

deploy-app-service.yml

Create a reusable GitHub workflow to deploy an Azure App Service using .NET, based on the structure of `deploy.yml`.

Building pipelines with Agents



app.py > ...

```python
from flask import Flask    Unable to import 'fl

app = Flask(__name__)

@app.route('/')
def home():    Missing function or method docstring
    return "Welcome to the Star Wars Collectibles Stor

if __name__ == '__main__':
    app.run(debug=True)    Final newline missing
```

```
s, colorama, blinker, Werkzeug, Jinja2, click, flask
Successfully installed Jinja2-3.1.6 MarkupSafe-3.0.2 W
erkzeug-3.1.3 blinker-1.9.0 click-8.1.8 colorama-0.4.6
 flask-3.1.0 itsdangerous-2.2.0

[notice] A new release of pip is available: 24.0 -> 25
.0.1
[notice] To update, run: python.exe -m pip install --u
pgrade pip
```

GitHub Copilot

✓ Generating plan to create a new workspace...

✓ Getting setup information...

I will create a Python-based project for your Star Wars collectibles store app. The project will use the latest version of Python and will be set up in the current workspace. Additionally, I will ensure the required Visual Studio Code extensions (`ms-python.python` and `ms-python.vscode-python-envs`) are installed.

Let me proceed with setting up the project.

✓ Python Extension for Visual Studio Code extension is already installed...

✓ Installing extension `Python Environments Extension for Visual Studio Code`...

The Python extension for Visual Studio Code is already installed. However, the Python Environments extension could not be installed. I will proceed with setting up the project and address any issues related to the missing extension later if necessary.

```
python -m venv venv && .\venv\Scripts\activate &&
pip install flask
```

The Python virtual environment has been successfully set up, and Flask has been installed as the web framework for the project. I will now create the initial project structure and files for the Star Wars collectibles store app.

3 files changed

- copilot-instructions.md .github
- app.py
- README.md

Demo

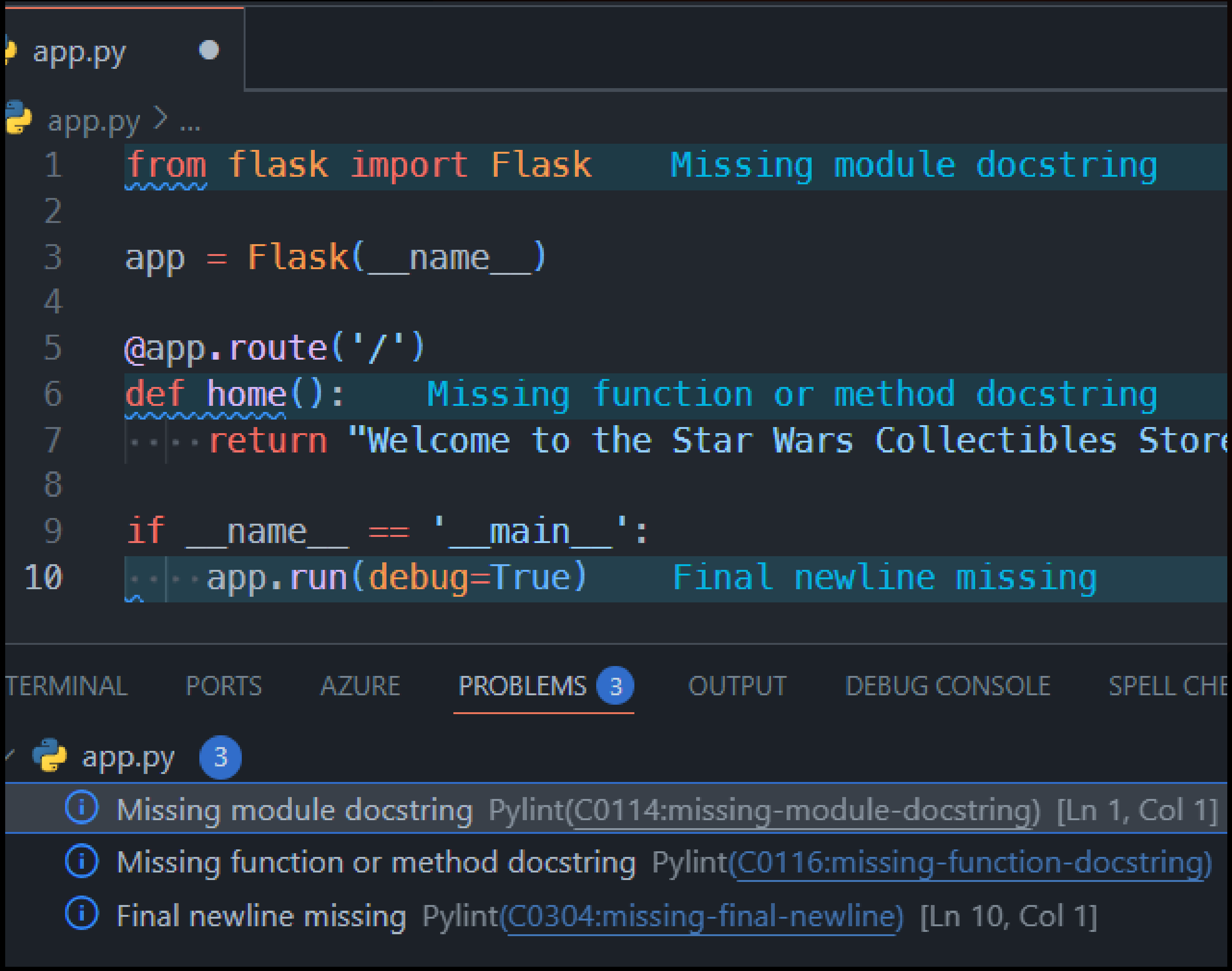
Let’s build an AI powered DevOps pipeline to deploy a web app to Azure.

Lint Tests

The Importance of Linting

What is Linting? A process that scans code to detect errors, enforce style guidelines, and improve quality before running tests or deploying.

Why is it Essential? It saves time by catching issues early, reduces bugs, and ensures smooth collaboration among developers.



```
from flask import Flask    Missing module docstring

app = Flask(__name__)

@app.route('/')
def home():    Missing function or method docstring
    return "Welcome to the Star Wars Collectibles Store

if __name__ == '__main__':
    app.run(debug=True)    Final newline missing
```

```
app.py 3
Missing module docstring  Pylint(C0114:missing-module-docstring)  [Ln 1, Col 1]
Missing function or method docstring  Pylint(C0116:missing-function-docstring)
Final newline missing  Pylint(C0304:missing-final-newline)  [Ln 10, Col 1]
```

Benefits of Linting

Improves Code Readability and Maintainability: Provides consistency in code structure. Clean, readable code fosters better collaboration within teams.

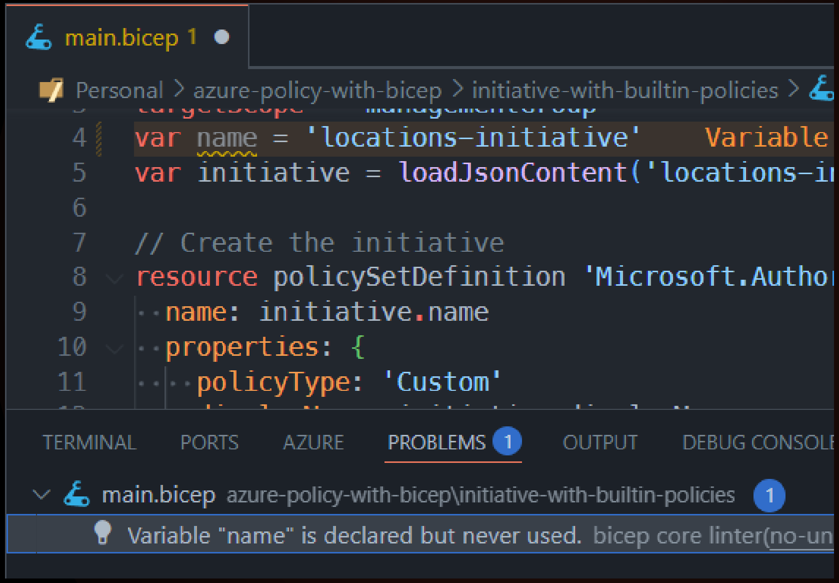
Detects Errors Before Runtime: Acts as a preventive measure, flagging syntax and logic errors before they escalate to runtime bugs.

Ensures Consistent Standards Across Teams: Linting aligns coding practices, reducing misunderstandings and inefficiencies during code reviews.

Promotes Best Practices: Encourages developers to adhere to guidelines that improve code, scalability, and security.

Linters

- Linters are tools designed to analyse code and align them to best practices.
- Linters are often added as extensions to IDEs or built into the IDE
- Linters can be installed as a standalone tools such as eslint, pylint, rubocop etc.



main.bicep 1

Personal > azure-policy-with-bicep > initiative-with-builtin-policies >

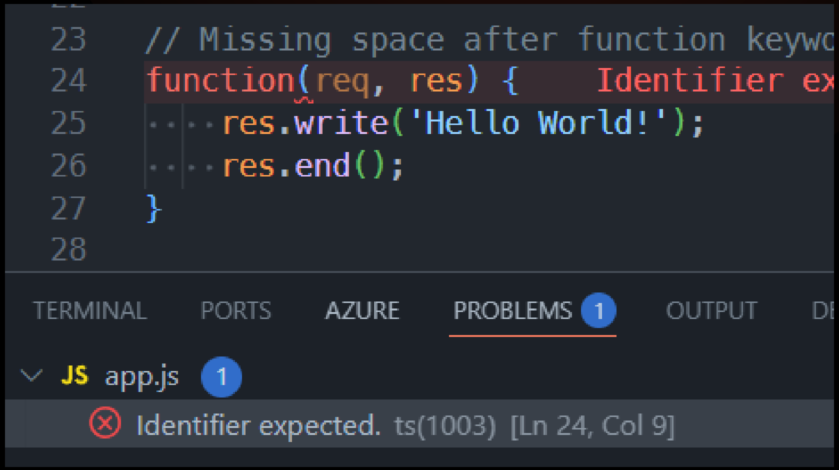
```
var name = 'locations-initiative'     Variable
var initiative = loadJsonContent('locations-i

// Create the initiative
resource policySetDefinition 'Microsoft.Autho
  name: initiative.name
  properties: {
    policyType: 'Custom'
```

TERMINAL PORTS AZURE PROBLEMS 1 OUTPUT DEBUG CONSOLE

main.bicep azure-policy-with-bicep\initiative-with-builtin-policies 1

Variable "name" is declared but never used. bicep core linter(no-un



```
// Missing space after function keywo
function(req, res) {    Identifier ex
    res.write('Hello World!');
    res.end();
}
```

TERMINAL PORTS AZURE PROBLEMS 1 OUTPUT DE

JS app.js 1

Identifier expected. ts(1003) [Ln 24, Col 9]

Demo

Let’s add lint tests to our DevOps pipeline.

Deploying AI Infrastructure and Models

Ingress

Private Networking: Implement private networking to ensure no public access, enhancing security and compliance

Ingress Flows with WAF: Support ingress flows through a Web Application Firewall (WAF) to protect against threats and manage traffic securely

Resiliency and Observability

Rate Limiting and Metrics Reporting: Implement rate limiting to control the usage and integrate logging into Azure Monitor and Log Analytics for detailed reporting on token usage and other critical metrics

Cost Control and Traceability: Monitor and trace consumption effectively to maintain cost control and ensure transparent tracking of resource usage

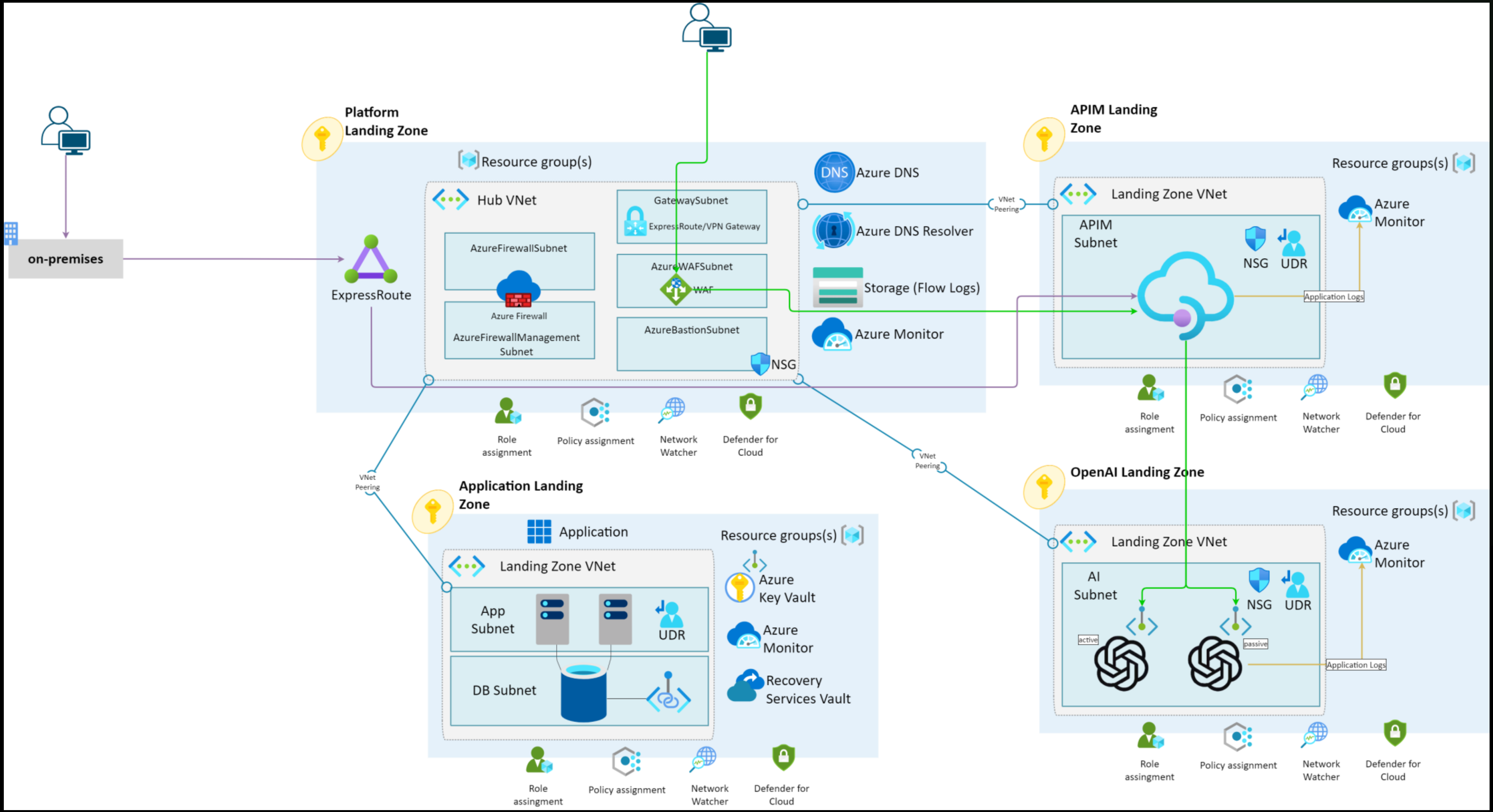
Resiliency: Ensure high availability and resiliency by deploying two AI instances in an active-passive setup, allowing for seamless failover when quotas are hit, or instances are unavailable

Architecture & Models

Shared Model: Deployed in a dedicated landing zone to support shared consumption across multiple services and ensure scalability for future growth

Distributed Model: Deployed directly within an application landing zone, ensuring that resources are tailored specifically to the needs of that individual application

Architecture



Demo

Let's explore how to handle AI model deployments with Azure Bicep for both shared and distributed scenarios.

PSRule for Azure

What is PSRule for Azure?

- An open-source tool for validating Azure resources against best practices
- Helps codify governance and shift-left compliance
- Based on Azure Well-Architected Framework & CAF

Why PSRule for Azure Matters

Automated compliance checks on ARM, Bicep, Terraform

Works in CI/CD pipelines (Azure DevOps, GitHub Actions)

Custom rules + out-of-the-box governance = faster decisions

Plug PSRule for Azure into DevOps Pipelines

Step 1	Add PSRule to pipeline (CLI or task)
Step 2	Validate templates pre-deploy
Step 3	Fail builds or notify on non-compliance

How can I use PSRule for Azure?

Detect overly permissive NSGs before deployment

Enforce naming conventions across all IaC

Catch missing diagnostic settings instantly

Demo

Let’s add PSRule for Azure to our DevOps pipeline.

Questions