Azure DevOps

Complete Project Guide

Ahmed Abdelwahed
ahmed@abdelwahed.me
www.abdelwahed.me
LinkedIn

Sprint 1

Azure Boards & Source Control Foundations

Plan your work, track your commits, control your code — before writing a single pipeline

Sprint 1 Learning Objectives

- Create an Azure DevOps organization and project with the Agile process template
- Provision the full Azure Boards work-item hierarchy using the Azure DevOps CLI
- Initialize a Git repository on Rocky Linux and push to Azure Repos
- Link Git commits to Work Items using the AB# notation
- Enforce branch policies: reviewers + linked work items
- Create and complete Pull Requests from the command line

Part 1 — Meet Your Environment

> **What we'll do:** Review the current server state and understand what we'll build over 5 days.

Component	Current State	What We Will Add
OS	Rocky Linux 10 (ro10)	Git + Azure DevOps Agent
Web Server	Nginx on port 8967	CI/CD auto-deploy target
Web App	Static portal: health, SSH, sudo, backup pages	Version-controlled in Azure Repos
Source Control	None — files on disk only	Local Git → Azure Repos
Deployment	Manual file edits on server	Automated pipeline with approval gate
Work Tracking	None — email/chat only	Azure Boards with linked commits

Part 2 — Provision Azure Boards with IaC

> **What we'll do:** Instead of clicking through the UI to create work items, we use the Azure DevOps CLI to provision the entire board hierarchy as code — repeatable and version-controlled.

2.1 Create the Azure DevOps Project

1. Go to https://dev.azure.com and sign in with your Microsoft account.
2. Click "+ New organization" if you don't have one → Name: Ahmed-App-Labs → Create.
3. In your organization, click "+ New project" and configure:

Name	Ahmed-App
Visibility	Private
Version control	Git
Work item process	Agile (gives Epics, Features, User Stories)

4. Click "Create project".

2.2 Install the Azure DevOps CLI Extension

Install the Azure DevOps CLI extension on ro10 — this lets you provision Boards work items from the command line instead of clicking through the portal.

```
# Install Azure CLI if not already installed
sudo dnf install -y azure-cli

# Add the Azure DevOps extension
az extension add --name azure-devops

# Log in to Azure
az login --use-device-code

# Set defaults so you don't repeat them in every command
az devops configure --defaults \
  organization=https://dev.azure.com/Ahmed-App-Labs \
  project=Ahmed-App-Linux
```

2.3 Provision the Full Work-Item Hierarchy

The script below creates the complete Epic → Feature → User Story hierarchy for all 5 days in one run.

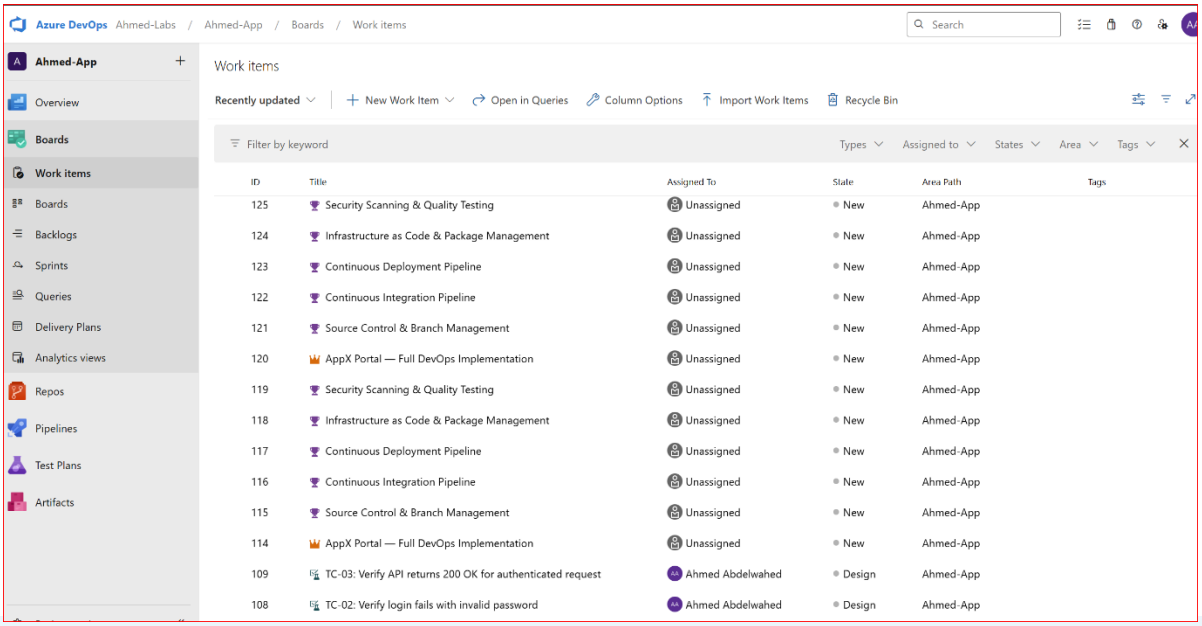
```
#deletes all work items from your Ahmed-App board cleanup-boards.sh
#!/bin/bash

ORG="https://dev.azure.com/Ahmed-Labs"
PROJECT="Ahmed-App"

az devops configure --defaults organization=$ORG project=$PROJECT

echo "Fetching ALL work items..."

IDS=$(az boards query \
  --wiql "SELECT [System.Id] FROM WorkItems WHERE [System.TeamProject] = '$PROJECT'" \
  --query "workItems[].id" -o tsv)

if [ -z "$IDS" ]; then
  echo "No work items found."
  exit 0
fi

echo "Deleting work items..."

for id in $IDS
do
  echo "Deleting ID: $id"
  az boards work-item delete --id $id --yes
done

echo "Done."
```

Azure DevOps Board Setup

```
#!/bin/bash
# ============================================================================
#  bootstrap-boards.sh
#  Ahmed-Labs | Ahmed-App — Azure Boards Bootstrap Script
# ----------------------------------------------------------------------------
#  Description : Creates all Epics and User Stories for the Ahmed-App board.
#                Safe to re-run — skips existing items, no duplicates.
#  Author      : Ahmed Abdelwahed
#  Contact     : ahmed@abdelwahed.me | www.abdelwahed.me
#  Course      : AZ-400 — DevOps Engineer Expert
#  Org         : https://dev.azure.com/Ahmed-Labs
#  Project     : Ahmed-App
# ----------------------------------------------------------------------------
#  Usage       : bash bootstrap-boards.sh
#  Requires    : Azure CLI + az devops extension + valid az login session
# ============================================================================

ORG="https://dev.azure.com/Ahmed-Labs"
PROJECT="Ahmed-App"
TAG="BOOTSTRAP-BOARD"

az devops configure --defaults organization=$ORG project=$PROJECT

echo "== Creating Epics =="
declare -A EPICS
EPICS["DevOps Foundation"]="Repo setup, branching strategy"
EPICS["CI/CD Platform"]="Build and deploy automation"
EPICS["Application Code"]="Portal UI, Node.js API, Docker"
EPICS["Cloud Deployment"]="Bicep IaC on Azure"
EPICS["Security & Secrets"]="Trivy scanning, Key Vault"
EPICS["Quality & Packages"]="Test Plans, Artifacts npm feed"
EPICS["Monitoring"]="App Insights, dashboards, alerts"

rm -f epic_ids.txt

for epic in "${!EPICS[@]}"
do
  id=$(az boards query \
    --wiql "SELECT [System.Id] FROM WorkItems
            WHERE [System.Title] = '$epic'
            AND [System.WorkItemType] = 'Epic'
            AND [System.Tags] CONTAINS '$TAG'" \
    --query "workItems[0].id" -o tsv)
  if [ -z "$id" ]; then
    id=$(az boards work-item create \
      --type "Epic" \
      --title "$epic" \
```

```
      --fields "System.Description=${EPICS[$epic]}" "System.Tags=$TAG" \
      --query id -o tsv)
    echo "Created Epic: $epic ($id)"
  else
    echo "Epic exists: $epic ($id)"
  fi
  echo "$epic=$id" >> epic_ids.txt
done

echo "== Creating Stories =="
declare -A STORIES
STORIES["Initialize Git repository"]="DevOps Foundation|2"
STORIES["Implement GitFlow branching strategy"]="DevOps Foundation|3"
STORIES["Create CI pipeline"]="CI/CD Platform|5"
STORIES["Create CD pipeline with approvals"]="CI/CD Platform|5"
STORIES["Build Nginx frontend portal"]="Application Code|3"
STORIES["Build Node.js API backend"]="Application Code|5"
STORIES["Dockerise both services"]="Application Code|3"
STORIES["Deploy to Azure using Bicep IaC"]="Cloud Deployment|5"
STORIES["Add Trivy security scanning to CI"]="Security & Secrets|3"
STORIES["Integrate Azure Key Vault for secrets"]="Security & Secrets|3"
STORIES["Set up Azure Artifacts npm feed"]="Quality & Packages|3"
STORIES["Create Azure Test Plan and run tests"]="Quality & Packages|3"
STORIES["Enable Application Insights monitoring"]="Monitoring|3"

declare -A EPIC_IDS
while IFS="=" read -r key value; do
  EPIC_IDS["$key"]=$value
done < epic_ids.txt

for title in "${!STORIES[@]}"
do
  IFS="|" read epic points <<< "${STORIES[$title]}"
  epic_id=${EPIC_IDS[$epic]}

  story_id=$(az boards query \
    --wiql "SELECT [System.Id] FROM WorkItems
            WHERE [System.Title] = '$title'
            AND [System.WorkItemType] = 'User Story'
            AND [System.Tags] CONTAINS '$TAG'" \
    --query "workItems[0].id" -o tsv)

  if [ -z "$story_id" ]; then
    story_id=$(az boards work-item create \
      --type "User Story" \
      --title "$title" \
```

```
      --fields "Microsoft.VSTS.Scheduling.StoryPoints=$points" "System.Tags=$TAG" \
      --query id -o tsv)
    echo "Created Story: $title ($story_id)"
  else
    echo "Story exists: $title ($story_id)"
  fi

  link_exists=$(az boards work-item show --id $story_id \
    --query "relations[?rel=='System.LinkTypes.Hierarchy-Reverse' && url.contains('$epic_id')]" -o
tsv)
  if [ -z "$link_exists" ]; then
    az boards work-item relation add \
      --id $story_id \
      --relation-type "Parent" \
      --target-id $epic_id
  fi
done

echo "== DONE: Ahmed-App board is ready. No duplicates, safe reruns =="
```



Why provision boards as code?

In real DevOps teams, even the project structure is version-controlled. Storing this script in your repo means:

- Any new team member can spin up the same boards in seconds
- Changes to work-item structure go through code review
- The script can be run in a pipeline to provision staging/test projects automatically

Part 3 — Git on Rocky Linux

> **What we'll do:** Install Git, configure identity, initialize the repository, and make the first commit — linking it to the Azure Board work item.

3.1 Install and Configure Git

Verify git is installed on ro10, then configure your global identity so every commit is attributed to you.

```
# Check if git is already installed
git --version

# Install git if not present
sudo dnf install -y git

# Configure your identity — every commit is permanently stamped with this
git config --global user.name "Your Name"
git config --global user.email "your@email.com"

# Verify configuration
git config --list
```

3.2 Initialize the Local Repository

Your web portal files live at /var/www/html/portal. We turn that directory into a Git repository.

Initialize a git repository inside the portal directory, creating the .git tracking folder.

```
cd /var/www/html/portal

# Initialize Git — creates a hidden .git folder
git init

# Check what files are present
ls -la
# Expected: index.html  health.html  ssh.html  sudo.html  backups.html

# Create .gitignore — exclude logs and temp files
echo "*.log"          >  .gitignore
echo "*.tmp"          >> .gitignore
echo ".backup_lock"   >> .gitignore

# Check repository status
git status
# Shows all portal files as "Untracked files" — ready to be committed
```

3.3 First Commit — Linked to Work Item

Notice the AB# syntax in the commit message — it automatically links this commit to the corresponding work item on Azure Boards.

Stage all portal files and create the first commit. The AB# tag in the message automatically links this commit to the work item on Azure Boards.

```
# Stage all files ("." means everything in this directory)
git add .

# Confirm what is staged
git status
# Shows: index.html  health.html  ssh.html  sudo.html  backups.html  .gitignore

# Commit with work item link — AB#7 auto-links to Azure Boards Story #7
git commit -m "feat: Initial commit Ahmed-App Linux Workshop Portal v1.0 AB#7"

# View commit history
git log --oneline
# Output: a7f3c21 feat: Initial commit Ahmed-App Linux Workshop Portal v1.0 AB#7
```

```
[ahmed@linux1 portal]$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
[ahmed@linux1 portal]$ ls
backups.html  health.html  index.html  ssh.html  sudo.html
[ahmed@linux1 portal]$ git pull origin master
remote: Azure Repos
remote: Found 4 objects to send. (10 ms)
Unpacking objects: 100% (4/4), 509 bytes | 509.00 KiB/s, done.
From https://dev.azure.com/Ahmed-Labs/Ahmed-App/_git/Ahmed-App
 * branch            master     -> FETCH_HEAD
   3ce0137..512424e  master     -> origin/master
Updating 3ce0137..512424e
Fast-forward
 index.html       | 3 ++-
 server_info.html | 5 +++++
 2 files changed, 7 insertions(+), 1 deletion(-)
 create mode 100755 server_info.html
[ahmed@linux1 portal]$ ls
backups.html  health.html  index.html  server_info.html  ssh.html  sudo.html
[ahmed@linux1 portal]$
```

3.4 Branching — Never Edit the Live Site Directly

> **Ahmed-App Problem**
>
> "We need to update health.html — but we can't break the live portal while editing."
> Solution: Feature branches. The live portal stays on master; all changes happen in isolation.

Create a feature branch for the new server_info.html page — never commit changes directly to master.

```
# Check which branch you are on
git branch

# Create and switch to a feature branch (linked to User Story #8)
git checkout -b feature/add-server-info-page

# Create a new server_info.html page
cat > /var/www/html/portal/server_info.html << EOF
<!DOCTYPE html>
<html><head><title>Server Info | Ahmed-App</title></head>
<body>
  <h1>Server Information</h1>
  <p>Hostname: ro10 | OS: Rocky Linux 10</p>
</body></html>
EOF

# Stage and commit — link to AB#8
git add server_info.html
git commit -m "feat: Add server info page to portal AB#8"

# Switch back to master — your new file is NOT here yet
git checkout master
ls /var/www/html/portal
# server_info.html is NOT listed — isolated safely in the branch

# Switch back to feature branch
git checkout feature/add-server-info-page

# Add nav link in index.html
nano index.html
# Add inside <nav>: <a href="server_info.html">Server Status</a>

git add index.html
git commit -m "feat: Add server info nav link AB#8"
```

Part 4 — Connect Rocky Linux to Azure Repos

> **What we'll do:** Authenticate ro10 with a Personal Access Token, push the code to Azure Repos, then enforce branch policies so no one can break master without review.

4.1 Create a Personal Access Token (PAT)

The PAT allows the Linux server to authenticate to Azure Repos without storing your Microsoft password.

1. In Azure DevOps, click your profile icon (top-right) → Personal access tokens.
2. Click "New Token" and configure:

Name	ro10-push-token
Expiration	90 days
Scopes	Code → Read & Write

3. Click "Create" — COPY THE TOKEN NOW. It will not be shown again.

Create a PAT in Azure DevOps, add Azure Repos as the remote origin, and push both the master and feature branches.

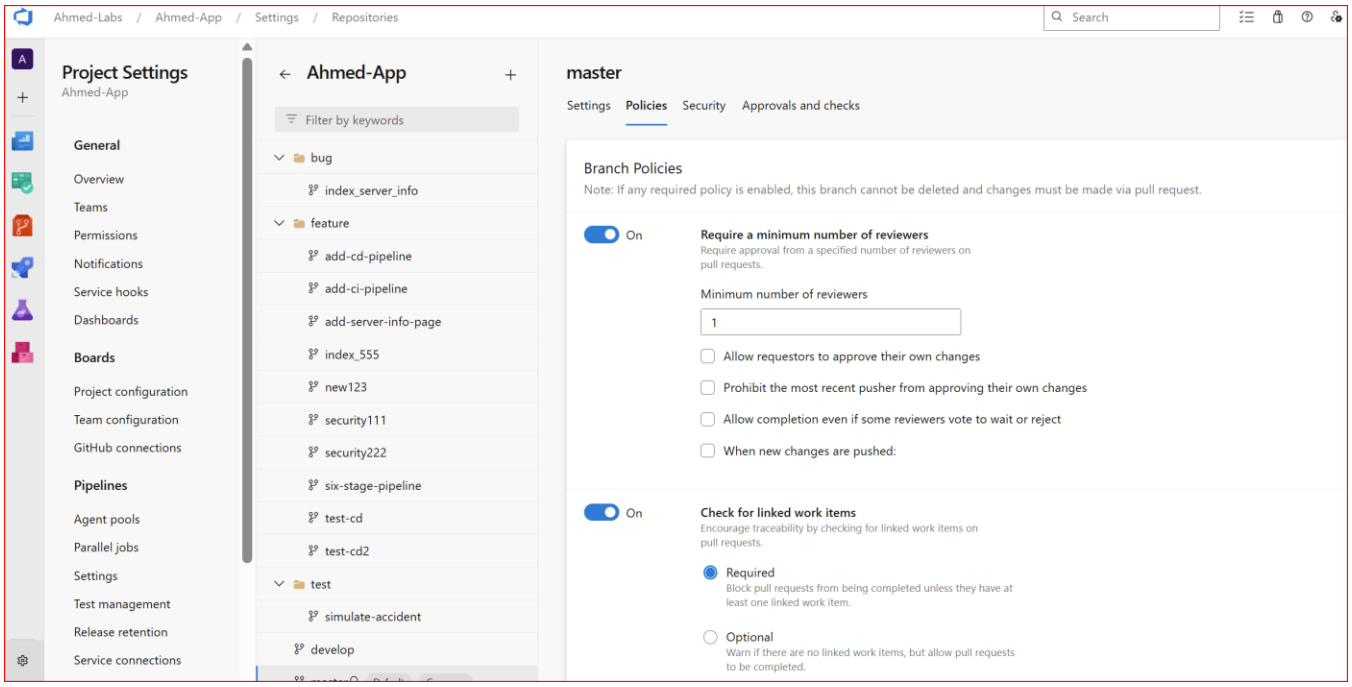
```
# On ro10 — inside /var/www/html/portal

# Add Azure Repos as the remote origin
git remote add origin \
  https://dev.azure.com/YourOrg/Ahmed-App-Linux/_git/Ahmed-App-Linux

# Verify remotes
git remote -v

# Store credentials so you only enter PAT once
git config --global credential.helper store

# Push master branch to Azure Repos
# When prompted for password: paste your PAT (NOT your Microsoft password)
git push -u origin master

# Push the feature branch too
git push origin feature/add-server-info-page
```

Verify: Go to Azure DevOps → Repos → Files. You should see all HTML files + .gitignore.

4.2 Enforce Branch Policies on Master

Branch policies prevent anyone (even you) from pushing broken code directly to master without review.

4. Go to Azure DevOps → Repos → Branches.
5. Click "..." next to master → Branch policies.
6. Enable: Minimum number of reviewers: 1.
7. Enable: Check for linked work items: Required.
8. Enable: Check for comment resolution: Required.
9. Leave Build Validation OFF for now — we add it in Day 2. Click Save.



Prove the branch protection is working — a direct push to master is blocked. The only path to master is through a PR.

```
# Test: Try to push directly to master — it will be BLOCKED
git checkout master
echo "<!-- direct push test -->" >> index.html
git add . && git commit -m "Direct push test"
git push origin master

# Error: GIT003: Push rejected. Branch requires a minimum of 1 reviewer.
# This is EXACTLY what we want — the policy works.

# Revert the test commit
git reset HEAD~1
git checkout -- index.html
```

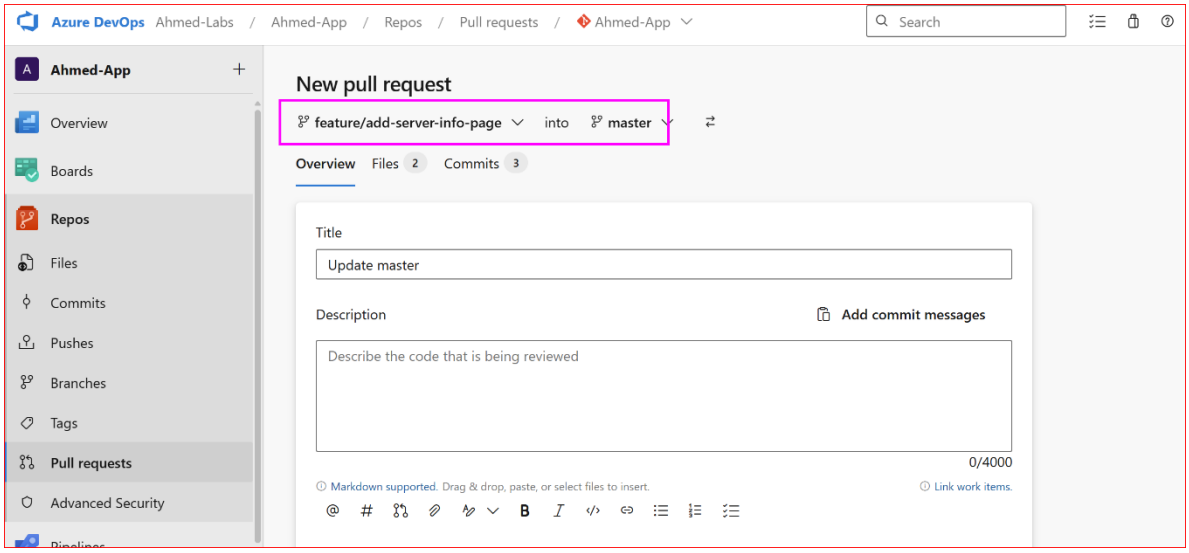
4.3 Complete the First Pull Request

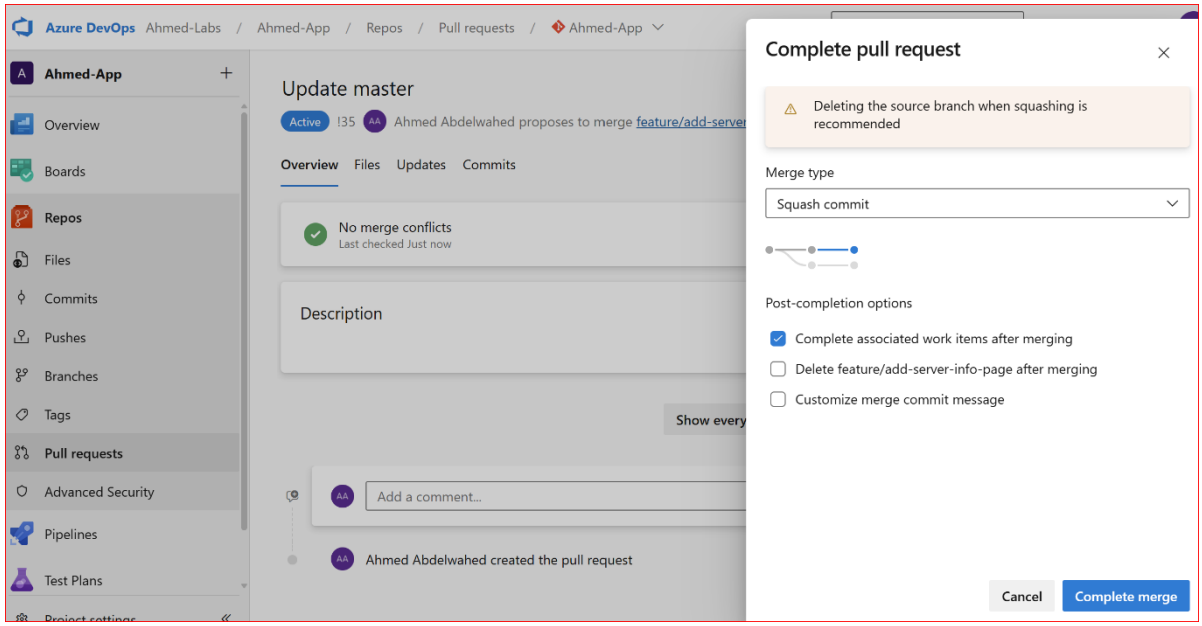
Push the feature branch to Azure Repos and open a Pull Request to merge it into master.

```
# Make sure you are on the feature branch with clean commits
git checkout feature/add-server-info-page
git log --oneline

# Push to Azure Repos
git push origin feature/add-server-info-page
```

10. Go to Azure DevOps → Repos → Pull Requests → New pull request.
11. Source: feature/add-server-info-page → Target: master.
12. Title: feat: Add server info page to Ahmed-App portal.
13. Link Work Items: click "+ Add work item" → link #8 (the User Story).
14. Reviewers: add a colleague (or your second account).
15. Review the Files tab — confirm server_info.html is added.
16. Add a comment: "Confirmed — HTML structure matches portal standard."

17. Resolve comment → Approve → Complete → select "Squash merge" → Complete merge.





After the PR is merged, sync your local master with the remote and verify the commit history is correct.

```
# After merge: pull master to sync your local copy
git checkout master
git pull origin master

# Verify: go to Azure Boards – Work Item #8 should move to "Resolved" automatically!
```

Why Pull Requests prevent outages

You just prevented the exact accident that caused the 3-hour outage:

- Branch Policy = rules that MUST be met before code enters master
- Pull Request = formal request to merge, with review and discussion
- AB#N in commit message = automatic work item linkage (traceability!)
- Squash merge = combines all commits into one clean entry on master

After this policy: even if you own the repo, you cannot push broken code to master solo.

Sprint 2

CI Pipeline — Continuous Integration

Make the computer check every change, so humans only approve — not inspect

Sprint 2 Learning Objectives

- Write a multi-step YAML CI pipeline triggered on HTML changes
- Install and configure a self-hosted Azure DevOps Agent on Rocky Linux 10
- Configure a pipeline agent pool and connect it to your project
- Add Build Validation policy to block broken PRs automatically
- Intentionally break the build to prove the quality gate works

Part 5 — Your First CI YAML Pipeline

What we'll do: Write a YAML pipeline that runs on every push, verifying file structure, nav links, and Nginx config — then publishes the portal files as a build artifact.

Ahmed-App Challenge

"The junior admin merged his fix through the PR process. But no automated check ran.
A reviewer approved it but nobody actually loaded the page to verify the fix worked.
We need the computer to validate every change before it can be merged.
If a navigation link is broken, the test must catch it — not a human."
Welcome to Continuous Integration.

5.1 Create the Pipeline File

Switch to the portal directory, pull latest master, and create a feature branch for the CI pipeline YAML file.

```
cd /var/www/html/portal
git checkout master && git pull origin master

# Create a feature branch for the pipeline
git checkout -b feature/add-ci-pipeline

# Create the pipeline YAML file
nano azure-pipelines.yml
```

5.2 CI Pipeline YAML — azure-pipelines.yml

Paste the following complete CI pipeline. Every step is explained with inline comments.

The basic CI pipeline YAML — validates file structure, nav links, Nginx config, and Bicep syntax, then publishes a build artifact on every push to master.

```
# Ahmed-App Portal — CI Pipeline
# Triggers on every push to master when HTML or pipeline files change
```

```
trigger:
  branches:
    include:
      - master
  paths:
    include:
      - "*.html"
      - "infra/*.bicep"
      - azure-pipelines.yml
      - azure-pipelines-cd.yml
    exclude:
      - "*.md"

pool:
  name: TEST_Agent  # Our self-hosted agent on ro10 (configured in Part 6)

variables:
  portalPath: "/var/www/html/portal"
  stagingPort: "8968"
  prodPort:    "8967"

steps:

# ─────────────────────────────────────────────────────────────
# STEP 1 — BUILD INFO
# ─────────────────────────────────────────────────────────────
- script: |
    echo "=== Ahmed-App Portal CI Build ==="
    echo "Branch  : $(Build.SourceBranchName)"
    echo "Commit  : $(Build.SourceVersion)"
    echo "By      : $(Build.RequestedFor)"
    echo "Build ID: $(Build.BuildId)"
  displayName: "Show Build Information"

# ─────────────────────────────────────────────────────────────
# STEP 2 — FILE STRUCTURE VALIDATION
# ─────────────────────────────────────────────────────────────
- script: |
    echo "Verifying portal file structure..."
    FAIL=0
    for page in index.html health.html ssh.html sudo.html backups.html server_info.html; do
      if [ ! -f "$page" ]; then
        echo "FAIL: $page is MISSING!"
        FAIL=1
      else
        echo "OK  : $page found"
      fi
    done
    [ $FAIL -eq 0 ] || exit 1
    echo "All portal pages verified."
  displayName: "Verify Portal File Structure"

# ─────────────────────────────────────────────────────────────
# STEP 3 — NAVIGATION LINK VALIDATION
# ─────────────────────────────────────────────────────────────
- script: |
    echo "Checking navigation links in index.html..."
    for page in health.html ssh.html sudo.html backups.html server_info.html; do
      grep -q "$page" index.html || { echo "FAIL: $page not linked in index.html"; exit 1; }
      echo "OK: $page is linked"
    done
    echo "All navigation links verified."
  displayName: "Validate Navigation Links"
```

```
# ─────────────────────────────────────────────────────────────
# STEP 4 — NGINX CONFIG VALIDATION
# ─────────────────────────────────────────────────────────────
- script: sudo nginx -t
  displayName: "Validate Nginx Config"

# ─────────────────────────────────────────────────────────────
# STEP 5 — BICEP TEMPLATE VALIDATION (added Day 4)
# ─────────────────────────────────────────────────────────────
- script: |
    if [ -f "infra/main.bicep" ]; then
      echo "Validating Bicep template..."
      az bicep build --file "$(Build.SourcesDirectory)/infra/main.bicep"
      echo "Bicep validation successful."
      rm -f "$(Build.SourcesDirectory)/infra/main.json"
    else
      echo "No Bicep file found — skipping validation."
    fi
  displayName: "Validate Bicep Template"

# ─────────────────────────────────────────────────────────────
# STEP 6 — PACKAGE AND PUBLISH ARTIFACT
# ─────────────────────────────────────────────────────────────
- task: CopyFiles@2
  inputs:
    sourceFolder: "$(Build.SourcesDirectory)"
    contents: |
      *.html
      infra/*.bicep
    targetFolder: "$(Build.ArtifactStagingDirectory)/portal"
  displayName: "Package Portal Files and Infra"

- publish: $(Build.ArtifactStagingDirectory)/portal
  artifact: Ahmed-App-portal
  displayName: "Publish Portal Artifact"
```

```
trigger:
  branches:
    include:
      - master          # Runs on every push to master
  paths:
    include:
      - "*.html"        # Only when HTML files change
      - azure-pipelines.yml
    exclude:
      - "*.md"          # Skip README-only changes

pool:
  name: TEST_Agent  # Our self-hosted agent on ro10 (configured in 5.2)
```

5.3 Commit and Push the Pipeline

Commit the CI pipeline YAML and push to the feature branch — link to Work Item #11 so the story auto-resolves on merge.

```
# Save the pipeline file and commit — link to Work Item #11
git add azure-pipelines.yml
git commit -m "feat: Add CI pipeline for Ahmed-App portal validation AB#11"
git push origin feature/add-ci-pipeline

# Create PR: feature/add-ci-pipeline → master
# Link Work Item: #11
# After merge, Story #11 auto-resolves
```

Option B — Upgrade CI with Comprehensive Smoke Tests

Once your staging server is running (port 8968), you can replace the basic CI pipeline with this upgraded version that adds 9 automated smoke test categories published to Azure Test Plans.

Smoke test categories covered:

1. HTTP Status Codes — every portal page returns HTTP 200
2. 404 Error Handling — non-existent pages return 404
3. Response Time — every page responds within 2 seconds
4. Content Validation — title tag, charset, file size checks
5. HTML Structure — DOCTYPE and closing html tag present
6. Security Headers — X-Frame-Options, X-Content-Type-Options, no Nginx version exposure
7. Content-Type — all pages return text/html
8. Navigation Links Live Check — all linked pages are reachable via HTTP
9. Sensitive Data Leak — scans HTML source for hardcoded passwords, API keys, tokens

Results are published as JUnit XML to Azure Test Plans automatically after every build.

5.4 Option B — Full Smoke Test CI Pipeline (azure-pipelines.yml)

This replaces the basic azure-pipelines.yml. Steps 1–5 (build info, file check, nav links, Nginx, Bicep) are identical. Step 6 adds the 9 smoke test categories. Step 7 publishes results to Test Plans.

Complete upgraded CI pipeline — copy this entire file into azure-pipelines.yml to replace the basic version. The smoke test step runs against http://localhost:8968 and publishes JUnit results to Azure Test Plans.

```
# ============================================================
# Ahmed-App Portal — CI Pipeline (with Comprehensive Smoke Tests)
# Agent   : TEST_Agent (Rocky Linux 10, self-hosted)
# Smoke Tests: HTTP status, 404, response time, content,
#              HTML structure, security headers, content-type,
#              nav links live check, sensitive data leak
# ============================================================

trigger:
  branches:
    include:
      - master
  paths:
    include:
      - '*.html'
      - 'infra/*.bicep'
      - azure-pipelines.yml
      - azure-pipelines-cd.yml
    exclude:
      - '*.md'

pool:
  name: TEST_Agent

variables:
  portalPath:  '/var/www/html/portal'
  stagingPort: '8968'
  prodPort:    '8967'
```

```
# ─────────────────────────────────────────────────────────────
# STEP 1 — BUILD INFO
# Print metadata about this run for traceability in the log
# ─────────────────────────────────────────────────────────────
steps:
- script: |
    echo "=== Ahmed-App Portal CI Build ==="
    echo "Branch:        $(Build.SourceBranchName)"
    echo "Commit:        $(Build.SourceVersion)"
    echo "Triggered by:  $(Build.RequestedFor)"
    echo "Build ID:      $(Build.BuildId)"
  displayName: 'Show Build Information'

# ─────────────────────────────────────────────────────────────
# STEP 2 — FILE STRUCTURE VALIDATION
# Fail the build immediately if any required portal file is missing
# ─────────────────────────────────────────────────────────────
- script: |
    echo "Verifying portal file structure..."
    FAIL=0
    for page in index.html health.html ssh.html sudo.html backups.html server_info.html; do
      if [ ! -f "$page" ]; then
        echo "FAIL: $page is MISSING!"
        FAIL=1
      else
        echo "OK: $page found"
      fi
    done
    [ $FAIL -eq 0 ] || exit 1
    echo "All portal pages verified."
  displayName: 'Verify Portal File Structure'

# ─────────────────────────────────────────────────────────────
# STEP 3 — NAVIGATION LINK VALIDATION
# Confirm every page is linked from index.html (static check)
# ─────────────────────────────────────────────────────────────
- script: |
    echo "Checking navigation links in index.html..."
    for page in health.html ssh.html sudo.html backups.html server_info.html; do
      grep -q "$page" index.html || { echo "FAIL: $page not linked in index.html"; exit 1; }
      echo "OK: $page is linked"
    done
    echo "All navigation links verified."
  displayName: 'Validate Navigation Links'

# ─────────────────────────────────────────────────────────────
# STEP 4 — NGINX CONFIG VALIDATION
# Run nginx -t to catch any config syntax errors before deploying
# ─────────────────────────────────────────────────────────────
- script: |
    echo "Validating Nginx configuration..."
    sudo nginx -t
  displayName: 'Validate Nginx Config'

# ─────────────────────────────────────────────────────────────
# STEP 5 — BICEP TEMPLATE VALIDATION
# Compile main.bicep to ARM JSON to catch syntax errors early
# ─────────────────────────────────────────────────────────────
- script: |
    echo "Validating Bicep template..."
    az bicep build --file "$(Build.SourcesDirectory)/infra/main.bicep"
    echo "Bicep validation successful."
```

```
    rm -f "$(Build.SourcesDirectory)/infra/main.json"
    echo "main.json removed."
  displayName: 'Validate Bicep Template'

# ──────────────────────────────────────────────────────────────
# STEP 6 — COMPREHENSIVE SMOKE TESTS
# 9 test categories run live against http://localhost:8968
# Results written to JUnit XML for Azure Test Plans
# ──────────────────────────────────────────────────────────────
- script: |
    echo "=== COMPREHENSIVE SMOKE TESTS — Ahmed-App Portal ==="
    OUTFILE="$(Build.ArtifactStagingDirectory)/smoke-results.xml"
    BASE="http://localhost:$(stagingPort)"
    PASS=0; FAIL=0; XML_CASES=""

    pass_test() {
      local ID="$1" DESC="$2"
      echo "  PASS [$ID] $DESC"
      PASS=$((PASS+1))
      XML_CASES="${XML_CASES}    <testcase classname="Ahmed-App.Smoke" name="${ID}: ${DESC}"/> "
    }

    fail_test() {
      local ID="$1" DESC="$2" MSG="$3"
      echo "  FAIL [$ID] $DESC — $MSG"
      FAIL=$((FAIL+1))
      XML_CASES="${XML_CASES}    <testcase classname="Ahmed-App.Smoke" name="${ID}: ${DESC}"> "
      XML_CASES="${XML_CASES}      <failure message="${MSG}"/> "
      XML_CASES="${XML_CASES}    </testcase> "
    }

    # ── 1. HTTP STATUS CODE TESTS ──────────────────────────
    echo "--- 1. HTTP Status Code Tests ---"
    for page in "" "health.html" "ssh.html" "sudo.html" "backups.html" "server_info.html"; do
      URL="$BASE/$page"; NAME="${page:-index.html}"
      CODE=$(curl -o /dev/null -s -w "%{http_code}" --max-time 10 "$URL" 2>/dev/null || echo "000")
      [ "$CODE" = "200" ] && pass_test "TC-HTTP-$NAME" "$NAME returns HTTP 200" \
                          || fail_test "TC-HTTP-$NAME" "$NAME returns HTTP 200" "Got HTTP $CODE"
    done

    # ── 2. 404 ERROR HANDLING ──────────────────────────────
    echo "--- 2. 404 Error Handling ---"
    CODE=$(curl -o /dev/null -s -w "%{http_code}" --max-time 10 "$BASE/notfound.html" 2>/dev/null ||
echo "000")
    [ "$CODE" = "404" ] && pass_test "TC-404" "Non-existent page returns 404" \
                        || fail_test "TC-404" "Non-existent page returns 404" "Got HTTP $CODE"

    # ── 3. RESPONSE TIME TESTS (threshold: 2s) ─────────────
    echo "--- 3. Response Time Tests ---"
    for page in "" "health.html" "ssh.html"; do
      URL="$BASE/$page"; NAME="${page:-index.html}"
      TIME=$(curl -o /dev/null -s -w "%{time_total}" --max-time 10 "$URL" 2>/dev/null || echo "99")
      SLOW=$(awk "BEGIN {print ($TIME > 2.0) ? 1 : 0}")
      [ "$SLOW" = "0" ] && pass_test "TC-PERF-$NAME" "$NAME responds within 2s (${TIME}s)" \
                        || fail_test "TC-PERF-$NAME" "$NAME responds within 2s" "${TIME}s exceeded
2s"
    done

    # ── 4. CONTENT VALIDATION ──────────────────────────────
    echo "--- 4. Content Validation Tests ---"
    for page in index.html health.html ssh.html sudo.html backups.html server_info.html; do
      grep -qi "<title>" "$page" 2>/dev/null \
        && pass_test "TC-TITLE-$page" "$page has title tag" \
```

```
      || fail_test "TC-TITLE-$page" "$page has title tag" "Missing title tag"
    SIZE=$(wc -c < "$page" 2>/dev/null || echo "0")
    [ "$SIZE" -gt 100 ] \
      && pass_test "TC-SIZE-$page" "$page not empty (${SIZE} bytes)" \
      || fail_test "TC-SIZE-$page" "$page not empty" "File size ${SIZE} bytes too small"
  done

  # ── 5. HTML STRUCTURE TESTS ──────────────────────────
  echo "--- 5. HTML Structure Tests ---"
  for page in index.html health.html ssh.html sudo.html backups.html server_info.html; do
    grep -qi "<!DOCTYPE" "$page" 2>/dev/null \
      && pass_test "TC-DOCTYPE-$page" "$page has DOCTYPE" \
      || fail_test "TC-DOCTYPE-$page" "$page has DOCTYPE" "Missing DOCTYPE"
    grep -qi "</html>" "$page" 2>/dev/null \
      && pass_test "TC-HTML-$page" "$page has closing html tag" \
      || fail_test "TC-HTML-$page" "$page has closing html tag" "Missing </html>"
  done

  # ── 6. SECURITY HEADERS ──────────────────────────────
  echo "--- 6. Security Headers Tests ---"
  HEADERS=$(curl -s -I --max-time 10 "$BASE/" 2>/dev/null)
  echo "$HEADERS" | grep -qi "x-frame-options" \
    && pass_test "TC-SEC-XFRAME" "X-Frame-Options header present" \
    || fail_test "TC-SEC-XFRAME" "X-Frame-Options header present" "Missing X-Frame-Options"
  echo "$HEADERS" | grep -qi "x-content-type-options" \
    && pass_test "TC-SEC-XCTYPE" "X-Content-Type-Options header present" \
    || fail_test "TC-SEC-XCTYPE" "X-Content-Type-Options header present" "Missing header"
  echo "$HEADERS" | grep -qi "server: nginx/" \
    && fail_test "TC-SEC-SERVER" "Nginx version not exposed" "Version found in Server header" \
    || pass_test "TC-SEC-SERVER" "Nginx version not exposed in Server header"

  # ── 7. CONTENT-TYPE TESTS ────────────────────────────
  echo "--- 7. Content-Type Tests ---"
  for page in "" "health.html" "ssh.html"; do
    URL="$BASE/$page"; NAME="${page:-index.html}"
    CTYPE=$(curl -o /dev/null -s -w "%{content_type}" --max-time 10 "$URL" 2>/dev/null || echo "")
    echo "$CTYPE" | grep -qi "text/html" \
      && pass_test "TC-CTYPE-$NAME" "$NAME returns Content-Type text/html" \
      || fail_test "TC-CTYPE-$NAME" "$NAME returns Content-Type text/html" "Got: $CTYPE"
  done

  # ── 8. NAVIGATION LINKS LIVE CHECK ──────────────────
  echo "--- 8. Navigation Links Live Check ---"
  for page in health.html ssh.html sudo.html backups.html server_info.html; do
    CODE=$(curl -o /dev/null -s -w "%{http_code}" --max-time 10 "$BASE/$page" 2>/dev/null || echo
"000")
    [ "$CODE" = "200" ] \
      && pass_test "TC-NAV-$page" "Linked page $page is reachable" \
      || fail_test "TC-NAV-$page" "Linked page $page is reachable" "Got HTTP $CODE"
  done

  # ── 9. SENSITIVE DATA LEAK TESTS ────────────────────
  # Scans HTML source for hardcoded credentials — complements Trivy
  echo "--- 9. Sensitive Data Leak Tests ---"
  PATTERNS=("password=" "passwd=" "secret=" "apikey=" "api_key=" "access_key=" "private_key="
"BEGIN RSA" "AKIAIO")
  for page in index.html health.html ssh.html sudo.html backups.html server_info.html; do
    PAGE_FAIL=0
    for pattern in "${PATTERNS[@]}"; do
      grep -qi "$pattern" "$page" 2>/dev/null && {
        fail_test "TC-LEAK-$page" "$page contains no sensitive data" "Pattern $pattern found"
        PAGE_FAIL=1
      }
```

```
      done
      [ "$PAGE_FAIL" = "0" ] && pass_test "TC-LEAK-$page" "$page contains no sensitive data patterns"
    done

    # ── Write JUnit XML ──────────────────────────────────
    TOTAL=$((PASS+FAIL))
    printf '%s\n' '<?xml version="1.0" encoding="UTF-8"?>' \
      "<testsuites name="Ahmed-App Smoke Tests" tests="${TOTAL}" failures="${FAIL}">" \
      "  <testsuite name="Ahmed-App.Smoke" tests="${TOTAL}" failures="${FAIL}">" \
      > "$OUTFILE"
    printf "%b" "$XML_CASES" >> "$OUTFILE"
    printf '%s\n' '  </testsuite>' '</testsuites>' >> "$OUTFILE"
    echo "Results: PASS=$PASS  FAIL=$FAIL  TOTAL=$TOTAL"
    echo "JUnit: $OUTFILE"
    [ $FAIL -eq 0 ] || exit 1
  displayName: 'Smoke Tests — 9 Categories'

# Publish JUnit XML results → Azure Test Plans (visible under Test Runs)
- task: PublishTestResults@2
  displayName: 'Publish Smoke Test Results to Test Plans'
  inputs:
    testResultsFormat: 'JUnit'
    testResultsFiles: '$(Build.ArtifactStagingDirectory)/smoke-results.xml'
    testRunTitle: 'Smoke Tests — Ahmed-App Portal CI'
    mergeTestResults: true
    failTaskOnFailedTests: true
  condition: always()

# ──────────────────────────────────────────────────────────
# STEP 7 — PACKAGE & PUBLISH ARTIFACT
# Copy portal HTML + Bicep into the artifact staging directory,
# then publish as the Ahmed-App-portal artifact for CD to consume
# ──────────────────────────────────────────────────────────
- task: CopyFiles@2
  inputs:
    sourceFolder: $(Build.SourcesDirectory)
    contents: |
      *.html
      infra/*.bicep
    targetFolder: $(Build.ArtifactStagingDirectory)/portal
  displayName: 'Package Portal Files and Infra'

- publish: $(Build.ArtifactStagingDirectory)/portal
  artifact: Ahmed-App-portal
  displayName: 'Publish Portal Artifact'
```

Part 6 — Self-Hosted Agent on Rocky Linux

What we'll do: Install and configure an Azure DevOps agent directly on ro10. This agent runs our pipeline steps on the actual server — giving the pipeline access to Nginx and the web root.

Why Self-Hosted?

Microsoft-hosted agents are clean VMs in Azure — great for generic builds.
But Ahmed-App needs to run "nginx -t" and deploy to /var/www/html/portal on ro10.
Only a self-hosted agent ON ro10 has access to the local Nginx and web root.

6.1 Create the Agent Pool in Azure DevOps

18. Go to Project Settings (bottom-left gear icon) → Agent pools.
19. Click "+ Add pool" → Type: Self-hosted | Name: TEST_Agent → Create.
20. Click "TEST_Agent" → "New agent" button → Linux → copy the download URL.

6.2 Install the Agent on ro10

Download and extract the Azure DevOps agent onto ro10. Use the URL copied from the Azure DevOps "New agent" dialog.

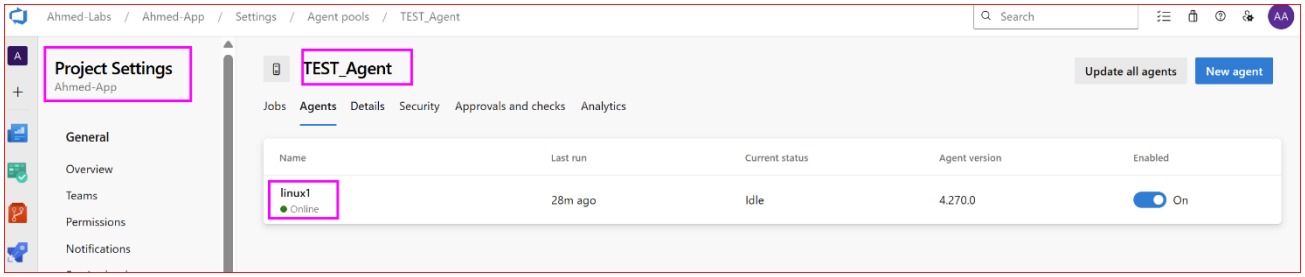
```
# On ro10 — run as your admin user (not root)

# Create a dedicated directory for the agent
mkdir -p ~/azagent && cd ~/azagent

# Download the latest agent (get URL from Azure DevOps "New agent" dialog)
wget https://vstsagentpackage.azureedge.net/agent/3.x.x/vsts-agent-linux-x64-3.x.x.tar.gz

# Extract
tar zxvf vsts-agent-linux-x64-*.tar.gz

# Configure the agent — follow the prompts
./config.sh
# Server URL  : https://dev.azure.com/YourOrg
# Auth type   : PAT
# PAT         : (paste your PAT — needs Agent Pools: Read & Manage scope)
# Agent pool  : TEST_Agent
# Agent name  : ro10-agent-01
# Work folder : _work (accept default)
```

Register the agent as a systemd service so it auto-starts on reboot, and grant it the sudo permissions needed to run nginx and copy files.

```
# Install agent as a systemd service (auto-starts on reboot)
sudo ./svc.sh install
sudo ./svc.sh start

# Verify the agent is running
sudo ./svc.sh status

# Grant the agent user sudo for nginx and file operations
# Add this to /etc/sudoers.d/azure-agent:
sudo visudo -f /etc/sudoers.d/azure-agent
# Add this line:
# ahmed ALL=(ALL) NOPASSWD: /usr/sbin/nginx, /usr/bin/systemctl reload nginx,
```

```
#          /usr/bin/cp, /usr/bin/mkdir, /usr/sbin/chcon, /usr/bin/chown

sudo chmod 0440 /etc/sudoers.d/azure-agent
```

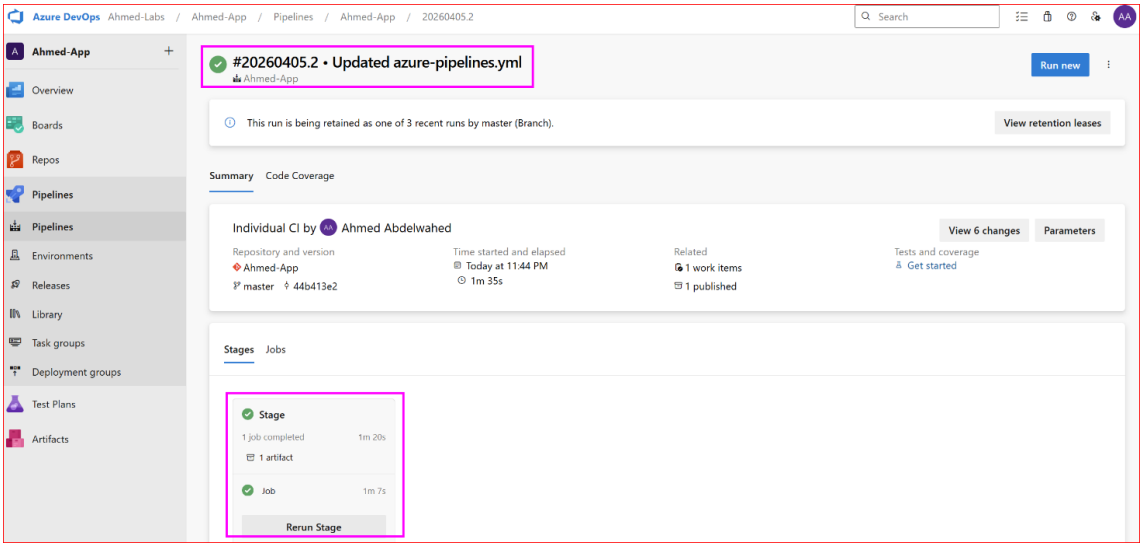
Verify: Go to Project Settings → Agent pools → TEST_Agent. Your agent "ro10-agent-01" should show as Online (green dot).



Azure DevOps — Agent online and ready in the TEST_Agent pool

6.3 Register and Run the Pipeline

21. In Azure DevOps → Pipelines → New pipeline.
22. Where is your code? → Azure Repos Git → Ahmed-App-Linux.
23. Configure: Existing Azure Pipelines YAML file.
24. Path: /azure-pipelines.yml → Continue → Run.
25. Watch all steps complete on YOUR ro10 server — each one maps to a commit you made.



Part 7 — Build Validation Policy & Quality Gates

> **What we'll do:** Add the CI pipeline as a required gate on pull requests. Then intentionally break the build to prove it catches the exact accident that caused the 3-hour outage.

7.1 Add Build Validation Policy to Master

26. Go to Repos → Branches → click "..." next to master → Branch policies.
27. Scroll to "Build Validation" section → click "+ Add build policy".

Field	Value
Build pipeline	Ahmed-App-Linux (the pipeline you just registered)
Trigger	Automatic (runs on every PR to master)
Policy requirement	Required (PR cannot complete if build fails)
Build expiration	Immediately when master is updated

28. Click Save. Mark Work Item #13 as Resolved.

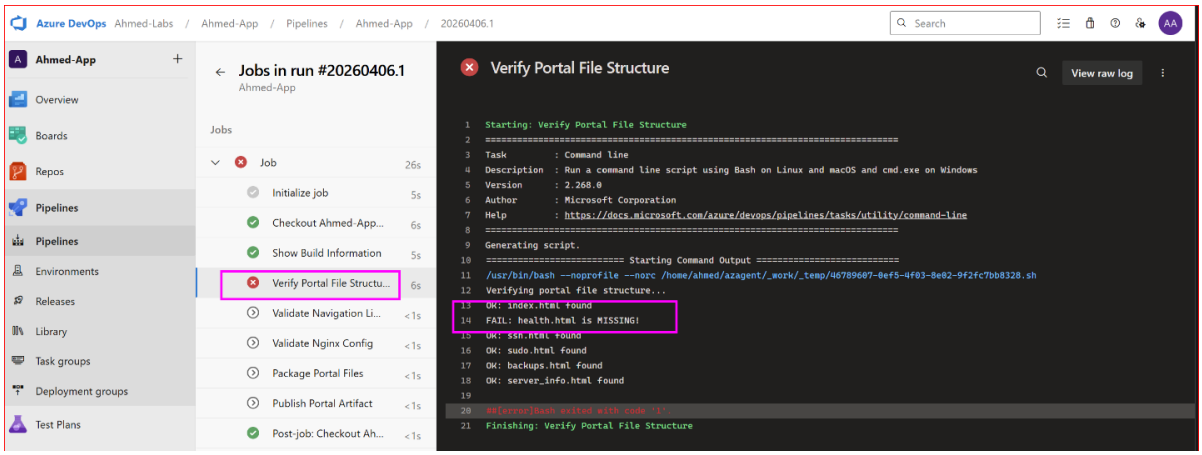
7.2 Break the Build on Purpose

Let's prove the pipeline catches the navigation link bug that caused the 3-hour outage.

Simulate the original accident — delete health.html and push to prove the pipeline catches the missing file before it can merge.

```
# Create a branch that simulates the original accident
git checkout master && git pull origin master
git checkout -b test/simulate-accident

# Simulate: delete health.html (the original accident)
rm health.html

# Commit and push the broken change
git add .
git commit -m "test: Simulate accidental deletion of health.html AB#14"
git push origin test/simulate-accident
```

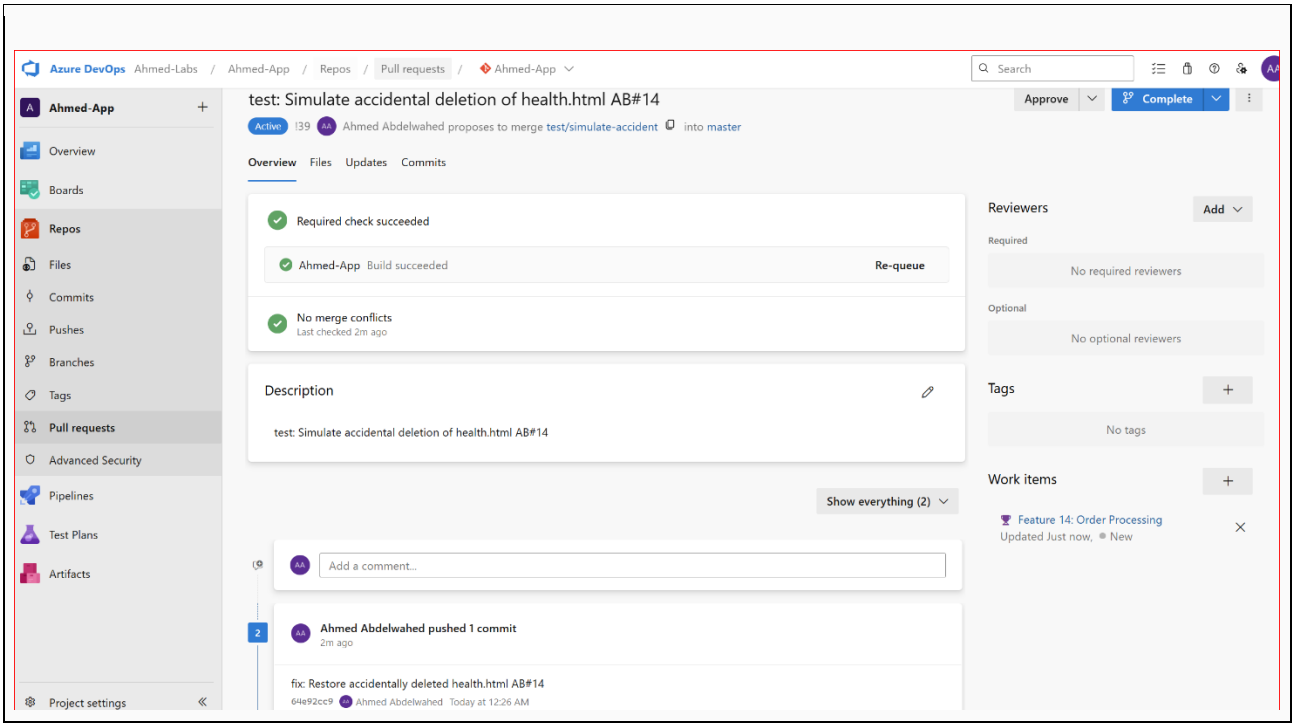
29. Create a PR: test/simulate-accident → master | Link Work Item #14.
30. Watch in Azure DevOps: Pipelines — the build FAILS at "Verify Portal File Structure".
31. Error: FAIL: health.html is MISSING!
32. The PR "Complete" button is GREYED OUT — the pipeline blocks the merge ✓



PR blocked — pipeline failed at "Verify Portal File Structure", Complete button greyed out

Restore the missing file from git history and push the fix — the pipeline re-runs automatically and the PR Complete button becomes active.

```
# Now fix it — restore health.html from git history
git checkout master -- health.html
git add health.html
git commit -m "fix: Restore accidentally deleted health.html AB#14"
git push origin test/simulate-accident

# Pipeline re-runs automatically
# → all steps pass → PR "Complete" button is active again
# → merge the PR → Story #14 resolves
```



Azure DevOps — Pipeline green after fix, PR Complete button active again

Sprint 3

CD Pipeline — Continuous Deployment

Automated deployment to Nginx — no tired engineer copying files at midnight

Sprint 3 Learning Objectives

- Create Azure DevOps Environments with approval gates
- Configure a staging Nginx server on port 8968 for safe pre-production testing
- Build a multi-stage CD YAML pipeline: Deploy Staging → Deploy Production
- Trigger a full deployment: approve in Azure DevOps → Nginx serves updated portal
- Link all commits and PRs to Work Items on the board

Part 8 — Configure Nginx Staging Server

> **What we'll do:** Set up a second Nginx virtual host on port 8968. This is our staging environment — the pipeline deploys here first, and you verify it works before approving production.

8.1 Create the Staging Nginx Config

Create the Nginx virtual host config file that serves the staging portal on port 8968.

```
# Create staging Nginx configuration
sudo nano /etc/nginx/conf.d/portal-staging.conf
```

Nginx server block for the staging environment — listens on port 8968, serves from portal-staging/, logs separately from production.

```
server {
    listen 8968;
    server_name _;
    root /var/www/html/portal-staging;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    # Log separately from production
    access_log /var/log/nginx/portal-staging-access.log;
    error_log  /var/log/nginx/portal-staging-error.log;
}
```

Create the staging web root directory, set SELinux context, open port 8968 in the firewall, and reload Nginx.

```
# Create staging directory with correct permissions
sudo mkdir -p /var/www/html/portal-staging
sudo chown -R $(whoami):$(whoami) /var/www/html/portal-staging
sudo chmod -R 755 /var/www/html/portal-staging
```

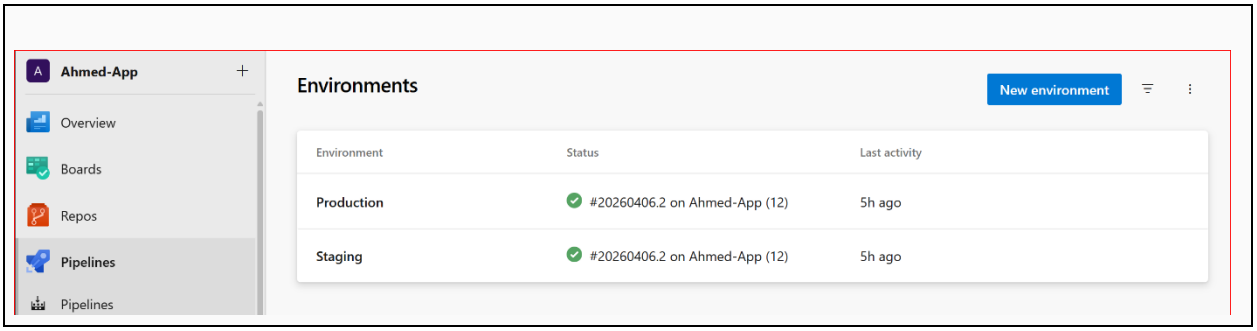
```
# Set SELinux context (Rocky Linux 10 requires this)
sudo chcon -t httpd_sys_content_t -R /var/www/html/portal-staging/

# Open port 8968 in firewall
sudo firewall-cmd --permanent --add-port=8968/tcp
sudo firewall-cmd --reload

# Allow SELinux to use port 8968 for HTTP
sudo semanage port -a -t http_port_t -p tcp 8968

# Test and reload Nginx
sudo nginx -t
sudo systemctl reload nginx

# Leave staging directory empty — the CD pipeline will populate it
```

Part 9 — Azure DevOps Environments & Approval Gates

What we'll do: Create two environments in Azure DevOps: Staging (auto-deploy) and Production (requires human approval). Production approval forces the team to verify staging before going live.

9.1 Create Deployment Environments

33. Go to Pipelines → Environments → "+ New environment".
34. Name: Staging | Resource: None → Create.
35. Create again → Name: Production | Resource: None → Create.
36. Click on Production environment → "..." → Approvals and checks.



Azure DevOps — Adding approval gate to the Production environment

37. Click "+" → Approvals → add yourself as approver.
38. Instructions: "Verify staging at http://ro10:8968 first, then approve."
39. Timeout: 24 hours → Click Create.

Environments and Approval Gates

Environments represent where your code runs and who approves deployments there:
- Staging = automatic deployment (fast feedback — see changes immediately after merge)
- Production = manual approval required (a human verifies staging is good before going live)

In real companies:
- Dev/Staging: auto-deploy on every merge to main
- Production: release manager approves, runs at scheduled maintenance window, with rollback plan

Part 10 — Multi-Stage CD Pipeline

> **What we'll do:** Write the CD pipeline YAML with two stages: Deploy Staging (automatic) and Deploy Production (waits for your approval). The CD pipeline downloads the artifact from the CI pipeline — never re-builds.

10.1 Create the CD Pipeline File

Create the feature branch and open the CD pipeline YAML for editing.

```
cd /var/www/html/portal
git checkout master && git pull origin master
git checkout -b feature/add-cd-pipeline
nano azure-pipelines-cd.yml
```

10.2 CD Pipeline YAML — azure-pipelines-cd.yml

The 2-stage CD pipeline YAML — Stage 1 deploys to local staging automatically, Stage 2 deploys to local production after your approval.

```
# Ahmed-App Portal — CD Pipeline
# Automatically starts after the CI pipeline ("Ahmed-App") succeeds

trigger: none  # CD never triggers directly — only via CI resource
pr: none

resources:
  pipelines:
    - pipeline: ciPipeline
      source: Ahmed-App      # Must match your CI pipeline name exactly
      trigger: true          # Runs automatically after CI succeeds

variables:
  portalPath:   "/var/www/html/portal"
  stagingPath:  "/var/www/html/portal-staging"
  nginxService: "nginx"

stages:

# ═══════════════════════════════════════════════════════
# STAGE 1: DEPLOY TO LOCAL STAGING (automatic — no approval)
# ═══════════════════════════════════════════════════════
- stage: DeployStaging
  displayName: "Deploy to Local Staging"
  jobs:
    - deployment: DeployStagingJob
      displayName: "Deploy Local Staging"
      environment: "Staging"
      pool:
        name: TEST_Agent
      strategy:
        runOnce:
          deploy:
            steps:
              - download: ciPipeline
                artifact: Ahmed-App-portal

              - script: |
                  echo "Deploying to LOCAL STAGING..."
                  sudo mkdir -p $(stagingPath)
```

```
                sudo cp $(Pipeline.Workspace)/ciPipeline/Ahmed-App-portal/*.html $(stagingPath)/
                sudo chcon -t httpd_sys_content_t -R $(stagingPath) || true
                echo "Validating staging..."
                curl -f http://localhost:8968/ || exit 1
                echo "Local staging deployment successful"
              displayName: "Deploy & Validate Local Staging"

# ============================================================
# STAGE 2: DEPLOY TO LOCAL PRODUCTION (requires approval)
# ============================================================
- stage: DeployProduction
  displayName: "Deploy to Local Production"
  dependsOn: DeployStaging
  condition: succeeded()
  jobs:
    - deployment: DeployProductionJob
      displayName: "Deploy Local Production"
      environment: "Production"      # ← approval gate is here
      pool:
        name: TEST_Agent
      strategy:
        runOnce:
          deploy:
            steps:
              - download: ciPipeline
                artifact: Ahmed-App-portal

              - script: |
                  echo "Creating backup..."
                  BACKUP_DIR="/backup/portal-$(Build.BuildId)"
                  sudo mkdir -p $BACKUP_DIR
                  sudo cp $(portalPath)/*.html $BACKUP_DIR/ 2>/dev/null || true
                displayName: "Backup Current Production"

              - script: |
                  echo "Deploying to LOCAL PRODUCTION..."
                  sudo cp $(Pipeline.Workspace)/ciPipeline/Ahmed-App-portal/*.html $(portalPath)/
                  sudo chcon -t httpd_sys_content_t -R $(portalPath) || true
                  sudo systemctl reload $(nginxService)
                  echo "Validating production..."
                  curl -f http://localhost:8967/ || exit 1
                  echo "Local production deployment successful"
                displayName: "Deploy & Validate Local Production"
```

10.3 Register and Test the CD Pipeline

Commit the 2-stage CD pipeline YAML, push to a feature branch, and register it as a new pipeline in Azure DevOps.

```
git add azure-pipelines-cd.yml
git commit -m "feat: Add CD pipeline with staging and production stages"
git push origin feature/add-cd-pipeline

# Create PR → merge → Story #17 resolves
```

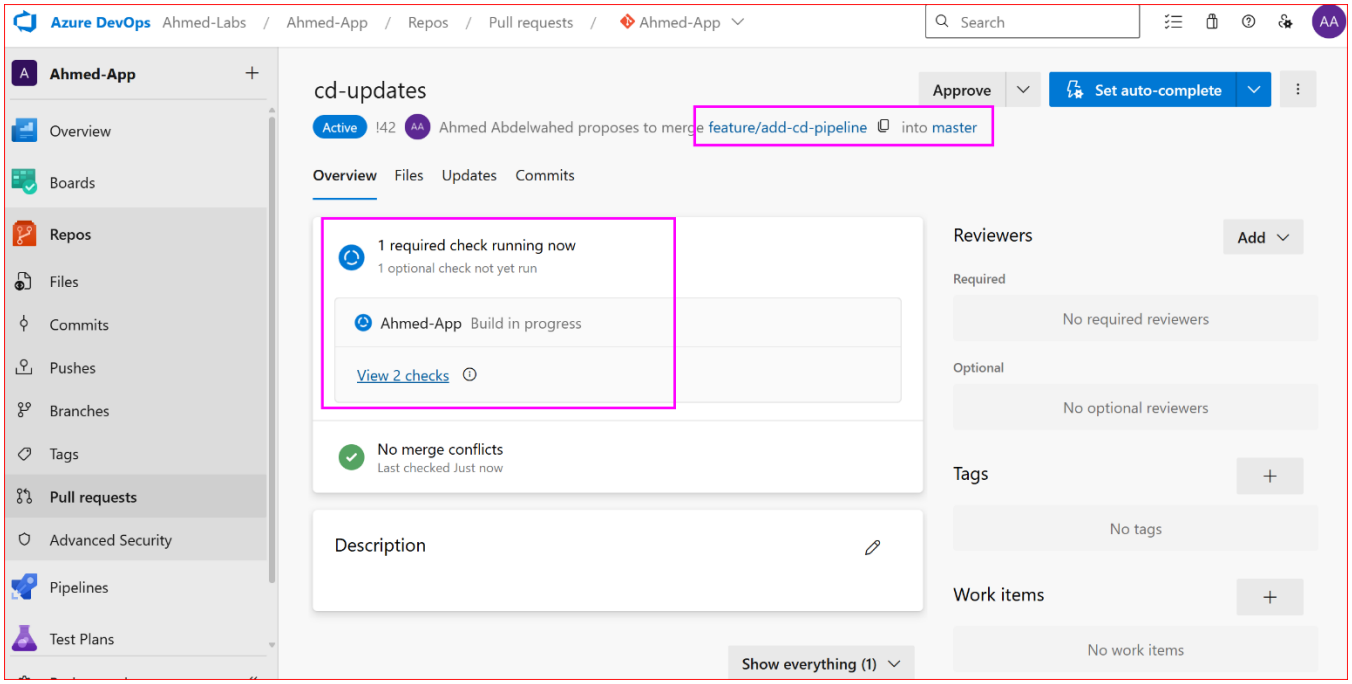
40. Go to Pipelines → New Pipeline → Azure Repos Git → Ahmed-App-Linux.
41. Choose: Existing YAML → select azure-pipelines-cd.yml.
42. Name the pipeline: Ahmed-App-CD → Save.

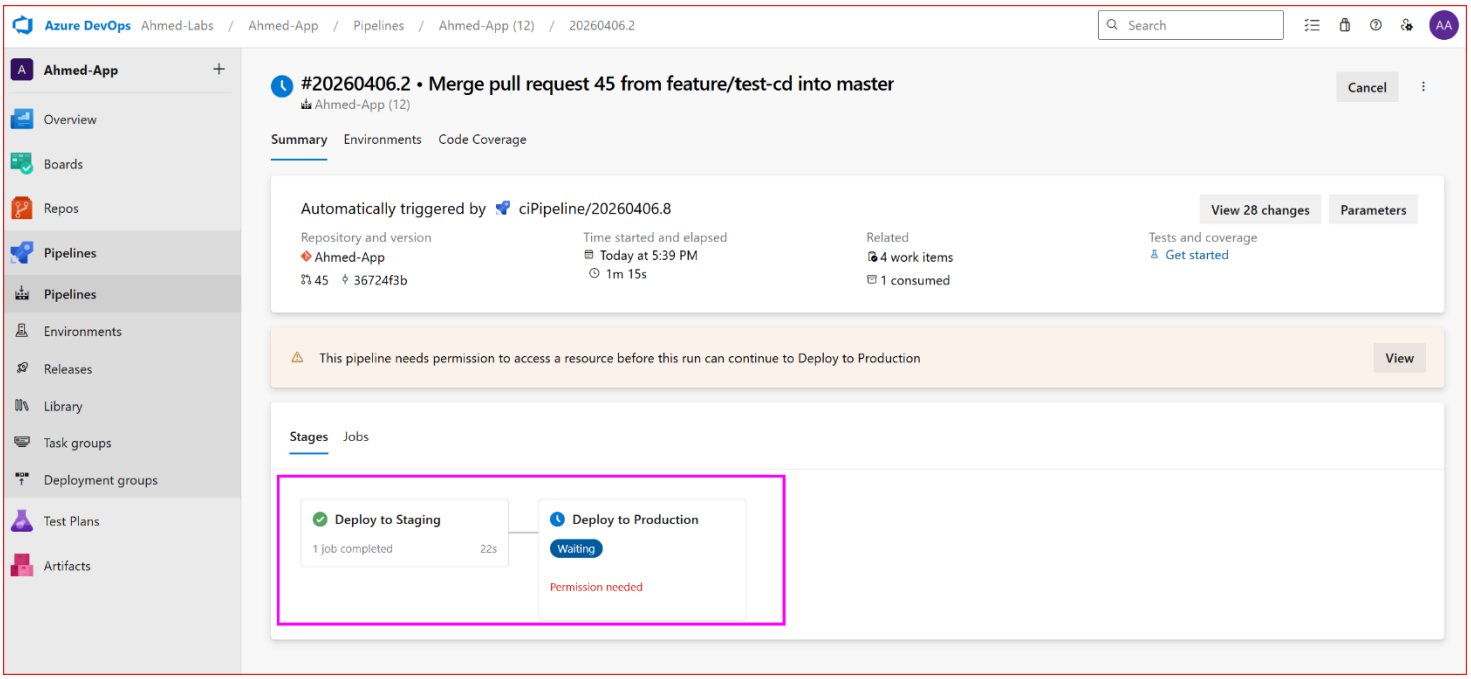
Watch the pipeline run:

- Deploy to Staging → green ✓ (automatic)

- Deploy to Production → ⏸ WAITING FOR APPROVAL

43. Verify staging first: curl http://localhost:8968/ (see updated portal).
44. In Azure DevOps: click the Production stage → "Review" → add comment: "Verified in staging — approving" → Approve.
45. Watch: Backup Current Production ✓ → Deploy to Production Nginx ✓ → HTTP status check: 200 ✓
46. Verify: curl http://localhost:8967/ — the updated portal is live!





Sprint 4

Infrastructure as Code with Bicep

Your infrastructure is code — version controlled, reviewed, deployed by the pipeline

Sprint 4 Learning Objectives

- Write and deploy a Bicep IaC file that creates Azure App Services
- Create an Azure Service Connection from Azure DevOps to your subscription
- Extend the pipeline to 5 stages including infrastructure deployment
- Implement GitFlow branching strategy (develop, release, hotfix branches)

> **Ahmed-App CTO Challenge**
>
> "Our backup server is in the same building as ro10. If the data centre loses power, both go dark. I want the portal on Azure — same code, same pipeline, zero extra maintenance."
>
> Your response: "We'll do it properly. No clicking in the Azure Portal to create resources. The infrastructure itself will be code — version controlled, reviewed, and deployed by the pipeline. Dev App Service first, human verifies, then Production App Service."
>
> CTO: "What is that called?" You: "Infrastructure as Code. Bicep. Let me show you."

Part 11 — Infrastructure as Code with Bicep

> **What we'll do:** Write a Bicep file that defines an App Service Plan, a Dev Web App, and a Prod Web App. The pipeline will deploy this infrastructure before deploying the portal code.

11.1 Install Bicep on ro10

Install the Azure CLI and the Bicep compiler on ro10.

```
# Install Azure CLI
sudo dnf install -y azure-cli

# Install Bicep extension
az bicep install

# Verify
az bicep version
```

11.2 Create the Bicep File — infra/main.bicep

Create the feature branch and the infra/ directory that will hold the Bicep file.

```
cd /var/www/html/portal
git checkout master && git pull origin master
git checkout -b feature/add-bicep-infrastructure

# Create infrastructure directory
mkdir -p infra
nano infra/main.bicep
```

The complete main.bicep file — defines an App Service Plan shared by a Dev and a Prod Web App. All names are derived from parameters so the file is fully reusable.

```
// Ahmed-App Portal – Azure Infrastructure
// Scope: Resource Group

// ── PARAMETERS ──────────────────────────────────────────
@description('Azure region for all resources')
param location string = 'canadacentral'

@description('Project name used in all resource names')
param projectName string = 'Ahmed-App-portal'

@description('Environment tag (dev | prod)')
@allowed(['dev', 'prod'])
param environment string = 'dev'

// ── VARIABLES ───────────────────────────────────────────
var planName    = 'plan-${projectName}'
var devAppName  = '${projectName}-dev'
var prodAppName = '${projectName}-prod'

// Common tags applied to every resource
var commonTags = {
  project:   projectName
  managedBy: 'bicep'
  environment: environment
}

// ── APP SERVICE PLAN ────────────────────────────────────
resource appServicePlan 'Microsoft.Web/serverfarms@2022-03-01' = {
  name:     planName
  location: location
  kind:     'linux'
  sku: {
    name: 'F1'
    tier: 'Free'
  }
  properties: {
    reserved: true  // Required for Linux App Service Plan
  }
  tags: commonTags
}

// ── DEV WEB APP ─────────────────────────────────────────
resource devWebApp 'Microsoft.Web/sites@2022-03-01' = {
  name:     devAppName
  location: location
  kind:     'app,linux'
  properties: {
    serverFarmId: appServicePlan.id
    httpsOnly: true
    siteConfig: {
      linuxFxVersion: 'NODE|20-lts'
      appSettings: [
        { name: 'ENVIRONMENT',    value: 'development' }
        { name: 'PORTAL_VERSION', value: '2.0' }
      ]
    }
  }
  tags: union(commonTags, { role: 'dev' })
}

// ── PROD WEB APP ────────────────────────────────────────
resource prodWebApp 'Microsoft.Web/sites@2022-03-01' = {
  name:     prodAppName
  location: location
  kind:     'app,linux'
  properties: {
```

```
    serverFarmId: appServicePlan.id
    httpsOnly: true
    siteConfig: {
      linuxFxVersion: 'NODE|20-lts'
      appSettings: [
        { name: 'ENVIRONMENT',    value: 'production' }
        { name: 'PORTAL_VERSION', value: '2.0' }
      ]
    }
  }
  tags: union(commonTags, { role: 'production' })
}

// ── OUTPUTS — used by the pipeline in later stages ──────────────
output devAppName  string = devWebApp.name
output prodAppName string = prodWebApp.name
output devUrl      string = 'https://${devWebApp.properties.defaultHostName}'
output prodUrl     string = 'https://${prodWebApp.properties.defaultHostName}'
output planName    string = appServicePlan.name
```

11.3 Validate Bicep Locally

Validate the Bicep file locally, then deploy the infrastructure to Azure using a what-if preview first.

```
# Compile Bicep to ARM to check for syntax errors
az bicep build --file infra/main.bicep

# Log in to Azure
az login --use-device-code
az account set --subscription "YourSubscriptionName"

# Create resource group
az group create --name rg-Ahmed-App-devops --location canadacentral

# What-if: preview what Azure WOULD create — without creating anything
az deployment group create --what-if \
  --resource-group rg-Ahmed-App-devops \
  --template-file infra/main.bicep
```

11.4 Commit and Push the Bicep File

Commit the Bicep file and open a PR to merge into master.

```
# Commit the Bicep file — link to Work Item #19
git add infra/main.bicep
git commit -m "feat: Add Bicep IaC for Azure App Service infrastructure AB#19"
git push origin feature/add-bicep-infrastructure

# Create PR → merge → Story #19 resolves
```

Bicep Key Concepts

- what-if: previews changes without applying them — always run before deploying
- Idempotent: run the same Bicep file 10 times — same result, no duplicates created
- Incremental mode (default): adds/updates, never deletes unlisted resources
- Complete mode: matches file exactly — deletes resources not in the template (use with caution!)
- param → var → resource → output is the standard Bicep file structure
- Outputs let the pipeline read values (URLs, names) from a completed deployment

Part 12 — Service Connection

What we'll do: Create an Azure Resource Manager service connection that gives the pipeline permission to deploy to your Azure subscription. One connection, used for both Bicep deployments and App Service deployments.

12.1 Create the Azure Service Connection

47. In Azure DevOps → Project Settings → Service connections → "+ New service connection".
48. Select: Azure Resource Manager → Next.
49. Authentication: Service principal (automatic) → Next.
50. Configure:

Scope level	Subscription
Subscription	Select your subscription from the dropdown
Resource group	rg-Ahmed-App-devops
Service connection name	azure-Ahmed-App-portal
Grant access to all pipelines	✓ Checked

51. Click Save → green tick confirms the connection is ready. Mark Work Item #20 as Resolved.

Service Principal — One Connection, Two Jobs

The Service Principal created here does two things in the pipeline:

1. Runs Bicep deployments (az deployment group create) in the Infrastructure stage
2. Deploys the app to both App Services (AzureWebApp@1) in the Azure Dev/Prod stages

Identity options:

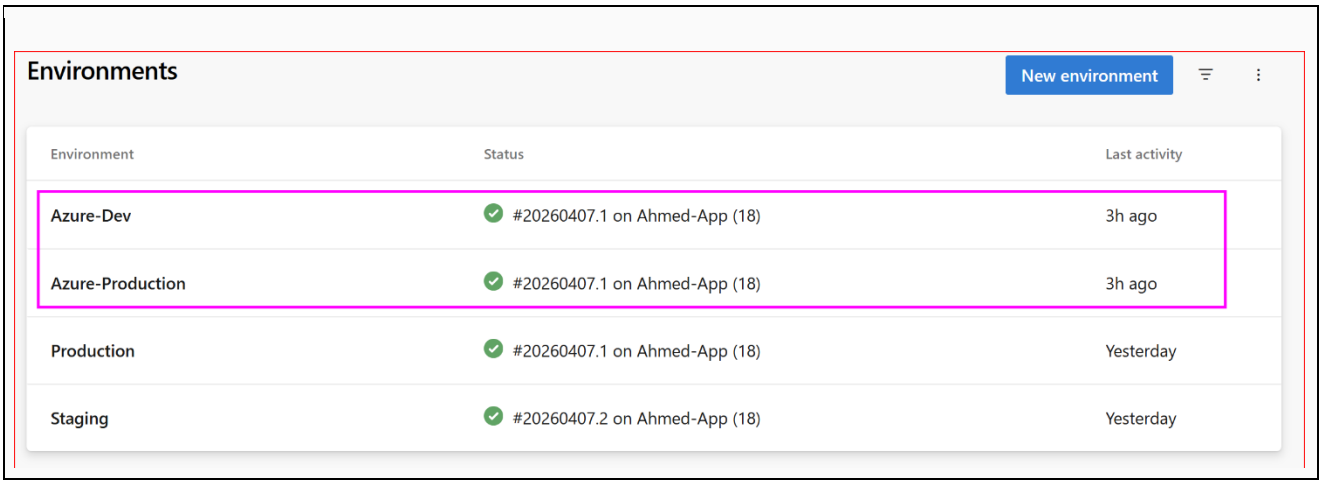
- Service Principal (classic) — client secret, must rotate periodically
- Workload Identity Federation — OIDC tokens, no secrets, recommended from 2024
- Managed Identity — for Azure resources calling other Azure resources

Part 13 — 5-Stage Pipeline with Bicep

> **What we'll do:** Extend the CD pipeline to 5 stages: Local Staging → Local Production → Deploy Azure Infrastructure (Bicep) → Azure Dev App Service → Azure Production App Service.

13.1 Create Two More Azure Environments

52. Go to Pipelines → Environments → "+ New environment".
53. Create: Name: Azure-Dev | Resource: None → Create.
54. Create: Name: Azure-Production | Resource: None → Create.
55. Add approval gate to Azure-Production: "..." → Approvals and checks → "+ Approvals" → add yourself.
56. Create: Name: DeployAzureInfra | Resource: None → Create (no approval needed — infrastructure only).



Azure DevOps — All 5 environments created with approval gates configured

Azure DevOps Environments — all 5 environments created with approval gates on Production and Azure-Production

13.2 Full 5-Stage CD Pipeline YAML

Update azure-pipelines-cd.yml with all 5 stages:

Create the feature branch for the 5-stage CD pipeline update.

```
git checkout master && git pull origin master
git checkout -b feature/5-stage-pipeline
nano azure-pipelines-cd.yml
```

The full 5-stage CD pipeline — Local Staging → Local Production → Azure Infrastructure (Bicep) → Azure Dev App Service → Azure Production App Service.

```
# Ahmed-App Portal — CD Pipeline (5 Stages)
# Auto-starts after CI pipeline succeeds

trigger: none
pr: none

resources:
  pipelines:
    - pipeline: ciPipeline
      source: "Ahmed-App"     # Match your CI pipeline name exactly
      trigger: true

variables:
```

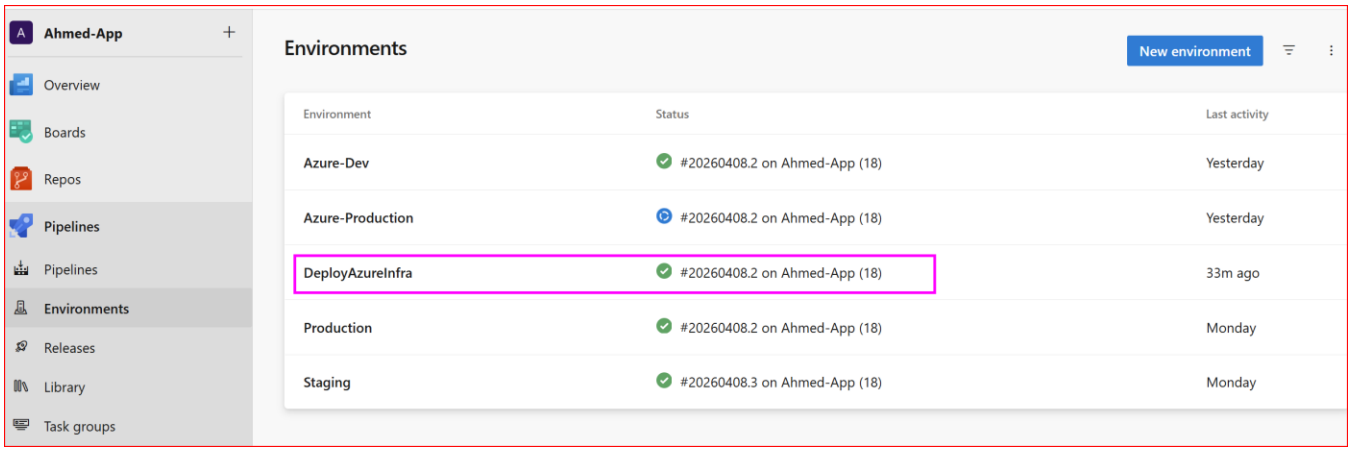
```yaml
  portalPath:        "/var/www/html/portal"
  stagingPath:       "/var/www/html/portal-staging"
  azureSubscription: "azure-Ahmed-App-portal"  # Service connection name
  resourceGroup:     "rg-Ahmed-App-devops"
  location:          "canadacentral"
  devAppName:        "Ahmed-App-portal-dev"
  prodAppName:       "Ahmed-App-portal-prod"

stages:

# ═══════════════════════════════════════════════════════════════
# STAGE 1: DEPLOY TO LOCAL STAGING (automatic)
# ═══════════════════════════════════════════════════════════════
- stage: DeployStaging
  displayName: "Deploy to Local Staging"
  jobs:
    - deployment: DeployStagingJob
      environment: "Staging"
      pool:
        name: TEST_Agent
      strategy:
        runOnce:
          deploy:
            steps:
              - download: ciPipeline
                artifact: Ahmed-App-portal
              - script: |
                  sudo mkdir -p $(stagingPath)
                  sudo cp $(Pipeline.Workspace)/ciPipeline/Ahmed-App-portal/*.html $(stagingPath)/
                  sudo chcon -t httpd_sys_content_t -R $(stagingPath) || true
                  curl -f http://localhost:8968/ || exit 1
                  echo "Local staging deployment successful"
                displayName: "Deploy & Validate Local Staging"

# ═══════════════════════════════════════════════════════════════
# STAGE 2: DEPLOY TO LOCAL PRODUCTION (requires approval)
# ═══════════════════════════════════════════════════════════════
- stage: DeployProduction
  displayName: "Deploy to Local Production"
  dependsOn: DeployStaging
  condition: succeeded()
  jobs:
    - deployment: DeployProductionJob
      environment: "Production"
      pool:
        name: TEST_Agent
      strategy:
        runOnce:
          deploy:
            steps:
              - download: ciPipeline
                artifact: Ahmed-App-portal
              - script: |
                  BACKUP_DIR="/backup/portal-$(Build.BuildId)"
                  sudo mkdir -p $BACKUP_DIR
                  sudo cp $(portalPath)/*.html $BACKUP_DIR/ 2>/dev/null || true
                  sudo cp $(Pipeline.Workspace)/ciPipeline/Ahmed-App-portal/*.html $(portalPath)/
                  sudo chcon -t httpd_sys_content_t -R $(portalPath) || true
                  sudo systemctl reload nginx
                  curl -f http://localhost:8967/ || exit 1
                  echo "Local production deployment successful"
                displayName: "Backup, Deploy & Validate Local Production"

# ═══════════════════════════════════════════════════════════════
```

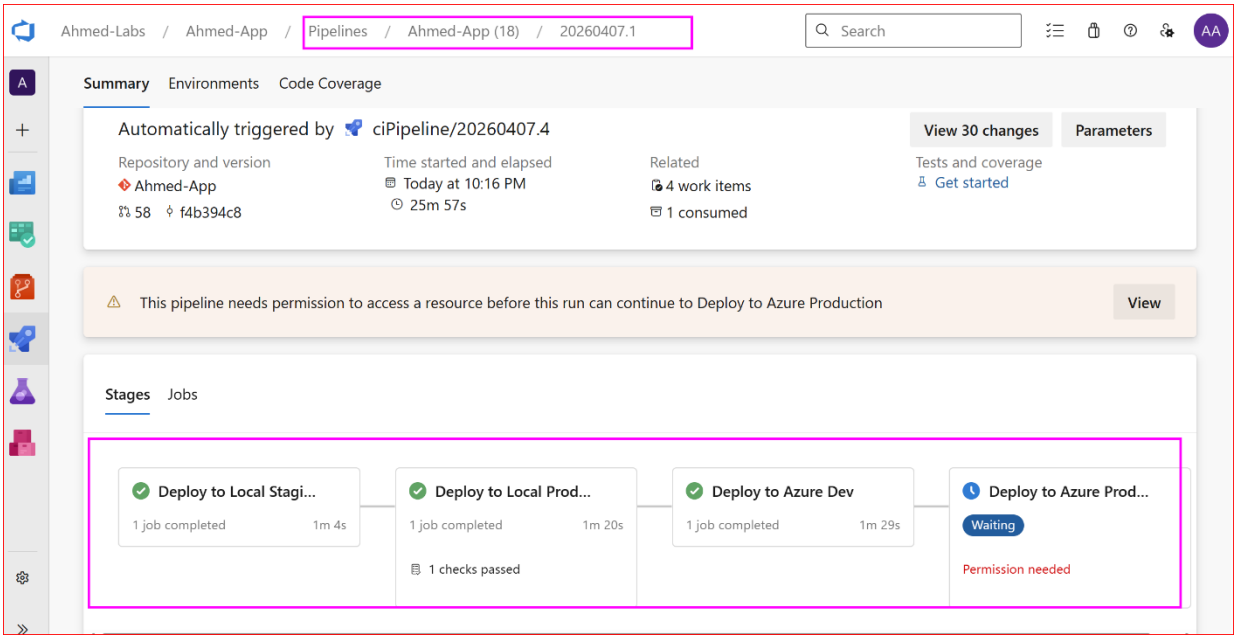
```
# STAGE 3: DEPLOY AZURE INFRASTRUCTURE FROM BICEP
# ═══════════════════════════════════════════════════════════════
- stage: DeployAzureInfra
  displayName: "Deploy Azure Infrastructure"
  dependsOn: DeployProduction
  condition: succeeded()
  jobs:
    - deployment: DeployAzureInfraJob
      environment: "DeployAzureInfra"
      pool:
        vmImage: "ubuntu-latest"
      strategy:
        runOnce:
          deploy:
            steps:
              - download: ciPipeline
                artifact: Ahmed-App-portal
              - task: AzureCLI@2
                displayName: "Deploy Azure Infrastructure from Bicep"
                inputs:
                  azureSubscription: "$(azureSubscription)"
                  scriptType: "bash"
                  scriptLocation: "inlineScript"
                  inlineScript: |
                    set -e
                    echo "Ensuring resource group exists..."
                    az group create --name $(resourceGroup) --location $(location)
                    echo "Deploying infrastructure from Bicep..."
                    az deployment group create \
                      --name Ahmed-App-infra-$(Build.BuildId) \
                      --resource-group $(resourceGroup) \
                      --template-file "$(Pipeline.Workspace)/ciPipeline/Ahmed-App-
portal/infra/main.bicep" \
                      --parameters location=$(location) projectName=Ahmed-App-portal environment=dev
                    echo "Infrastructure deployment completed."

# ═══════════════════════════════════════════════════════════════
# STAGE 4: DEPLOY TO AZURE DEV APP SERVICE (automatic)
# ═══════════════════════════════════════════════════════════════
- stage: DeployAzureDev
  displayName: "Deploy to Azure Dev"
  dependsOn: DeployAzureInfra
  condition: succeeded()
  jobs:
    - deployment: DeployAzureDevJob
      environment: "Azure-Dev"
      pool:
        vmImage: "ubuntu-latest"
      strategy:
        runOnce:
          deploy:
            steps:
              - download: ciPipeline
                artifact: Ahmed-App-portal
              - script: |
                  mkdir -p $(Pipeline.Workspace)/webroot
                  cp $(Pipeline.Workspace)/ciPipeline/Ahmed-App-portal/*.html
$(Pipeline.Workspace)/webroot/
                displayName: "Prepare Webroot Files"
              - task: ArchiveFiles@2
                displayName: "Create Deployment ZIP"
                inputs:
                  rootFolderOrFile: "$(Pipeline.Workspace)/webroot"
                  includeRootFolder: false
```

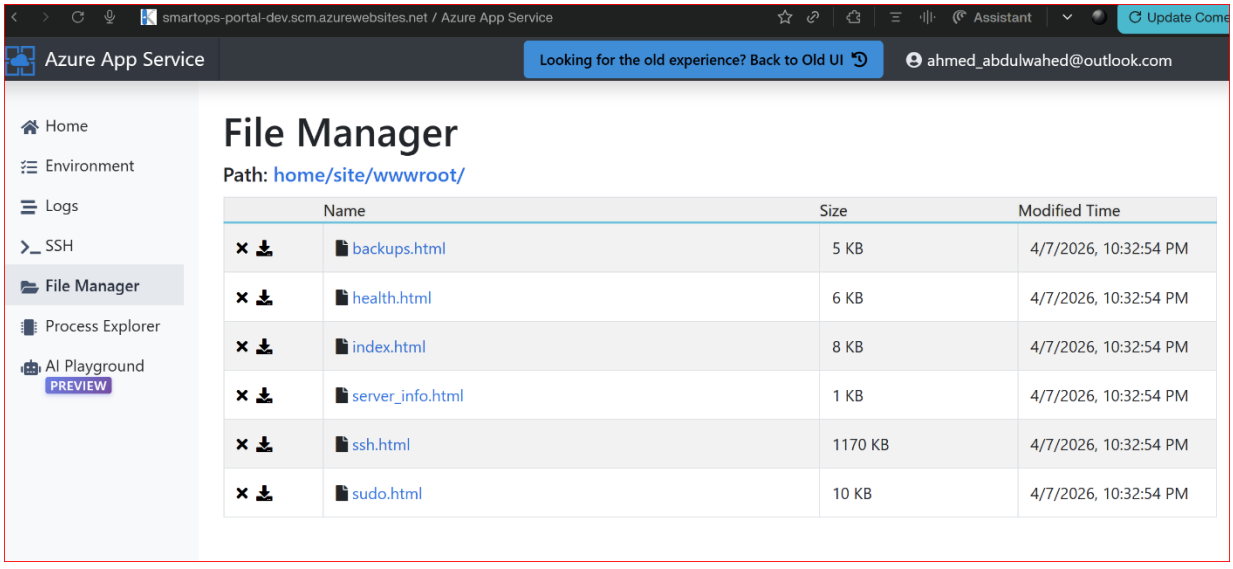
```
                archiveType: zip
                archiveFile: "$(Pipeline.Workspace)/portal.zip"
            - task: AzureWebApp@1
              displayName: "Deploy to Azure Dev Web App"
              inputs:
                azureSubscription: "$(azureSubscription)"
                appType: webAppLinux
                appName: "$(devAppName)"
                package: "$(Pipeline.Workspace)/portal.zip"
            - script: |
                sleep 20
                curl -f https://$(devAppName).azurewebsites.net/ || exit 1
                echo "Azure Dev deployment successful"
              displayName: "Validate Azure Dev"

# ═══════════════════════════════════════════════════════════════
# STAGE 5: DEPLOY TO AZURE PRODUCTION (requires approval)
# ═══════════════════════════════════════════════════════════════
- stage: DeployAzureProd
  displayName: "Deploy to Azure Production"
  dependsOn: DeployAzureDev
  condition: succeeded()
  jobs:
    - deployment: DeployAzureProdJob
      environment: "Azure-Production"
      pool:
        vmImage: "ubuntu-latest"
      strategy:
        runOnce:
          deploy:
            steps:
              - download: ciPipeline
                artifact: Ahmed-App-portal
              - script: |
                  mkdir -p $(Pipeline.Workspace)/webroot
                  cp $(Pipeline.Workspace)/ciPipeline/Ahmed-App-portal/*.html
$(Pipeline.Workspace)/webroot/
                displayName: "Prepare Webroot Files"
              - task: ArchiveFiles@2
                displayName: "Create Production ZIP"
                inputs:
                  rootFolderOrFile: "$(Pipeline.Workspace)/webroot"
                  includeRootFolder: false
                  archiveType: zip
                  archiveFile: "$(Pipeline.Workspace)/portal.zip"
              - task: AzureWebApp@1
                displayName: "Deploy to Azure Production Web App"
                inputs:
                  azureSubscription: "$(azureSubscription)"
                  appType: webAppLinux
                  appName: "$(prodAppName)"
                  package: "$(Pipeline.Workspace)/portal.zip"
              - script: |
                  sleep 20
                  curl -f https://$(prodAppName).azurewebsites.net/ || exit 1
                  echo "Azure Production deployment successful"
                displayName: "Validate Azure Production"
```

Commit the 5-stage CD pipeline — merge triggers the full chain: Local Staging → Local Production → Azure Infra → Azure Dev → Azure Production.

```
git add azure-pipelines-cd.yml
git commit -m "feat: Add 5-stage pipeline with Bicep IaC and Azure App Service deploy AB#21"
git push origin feature/5-stage-pipeline
# PR → merge → watch all 5 stages complete!
```


Ahmed-App
Overview
Boards
Repos
Pipelines
Pipelines
Environments
Releases
Library
Task groups
Environments
New environment
Environment
Status
Last activity
Azure-Dev
#20260408.2 on Ahmed-App (18)
Yesterday
Azure-Production
#20260408.2 on Ahmed-App (18)
Yesterday
DeployAzureInfra
#20260408.2 on Ahmed-App (18)
33m ago
Production
#20260408.2 on Ahmed-App (18)
Monday
Staging
#20260408.3 on Ahmed-App (18)
Monday


Ahmed-Labs / Ahmed-App / Pipelines / Ahmed-App (18) / 20260407.1
Search
AA
Summary
Environments
Code Coverage
Automatically triggered by ciPipeline/20260407.4
View 30 changes
Parameters
Repository and version
Ahmed-App
58 f4b394c8
Time started and elapsed
Today at 10:16 PM
25m 57s
Related
4 work items
1 consumed
Tests and coverage
Get started
This pipeline needs permission to access a resource before this run can continue to Deploy to Azure Production
View
Stages
Jobs
Deploy to Local Stagi...
1 job completed
1m 4s
Deploy to Local Prod...
1 job completed
1m 20s
1 checks passed
Deploy to Azure Dev
1 job completed
1m 29s
Deploy to Azure Prod...
Waiting
Permission needed


smartops-portal-dev.scm.azurewebsites.net / Azure App Service
Assistant
Update Come
Azure App Service
Looking for the old experience? Back to Old UI
ahmed_abdulwahed@outlook.com
Home
Environment
Logs
SSH
File Manager
Process Explorer
AI Playground
PREVIEW
File Manager
Path: home/site/wwwroot/
Name
Size
Modified Time
backups.html
5 KB
4/7/2026, 10:32:54 PM
health.html
6 KB
4/7/2026, 10:32:54 PM
index.html
8 KB
4/7/2026, 10:32:54 PM
server_info.html
1 KB
4/7/2026, 10:32:54 PM
ssh.html
1170 KB
4/7/2026, 10:32:54 PM
sudo.html
10 KB
4/7/2026, 10:32:54 PM

Sprint 5

Security, Testing & Monitoring

Scan for vulnerabilities, store secrets safely, test everything, observe always

Sprint 5 Learning Objectives

- Integrate Trivy security scanner into the CI pipeline to block vulnerable builds
- Store secrets in Azure Key Vault and consume them safely in the pipeline
- Create Azure Test Plans with manual and automated test cases
- Link test results to Work Items in Azure Boards
- Set up Application Insights for live portal monitoring and alerting
- Close all work items and complete Sprint 1

Part 16 — Security Scanning with Trivy

What we'll do: Install Trivy on ro10 and add two security scan steps to the CI pipeline: one for file system vulnerabilities (CVEs) and one for hardcoded secrets. If either fails, the pipeline blocks the build.

Ahmed-App Security Audit

"The security team ran an audit. Our portal files include a vendor JavaScript library with a known CVE.
If we had deployed this to production before catching it, we would have had a data breach notification to file.
We need automated security scanning in the pipeline."

16.1 Install Trivy on ro10

Add the Trivy repository and install the Trivy vulnerability scanner on ro10.

```
# Import Trivy GPG key and add repository
sudo rpm --import https://aquasecurity.github.io/trivy-repo/rpm/public.key

cat << EOF | sudo tee /etc/yum.repos.d/trivy.repo
[trivy]
name=Trivy repository
baseurl=https://aquasecurity.github.io/trivy-repo/rpm/releases/$basearch/
gpgcheck=0
enabled=1
gpgkey=https://aquasecurity.github.io/trivy-repo/rpm/public.key
EOF

sudo dnf install -y trivy

# Verify
trivy --version
```

```
# Download the vulnerability database (takes 1-2 minutes first time)
trivy image --download-db-only
```

16.2 Test Trivy Locally

Test Trivy locally — scans for HIGH/CRITICAL CVEs and hardcoded secrets before adding it to the pipeline.

```
cd /var/www/html/portal

# Scan portal directory for HIGH and CRITICAL vulnerabilities
trivy fs . \
  --severity HIGH,CRITICAL \
  --exit-code 1 \
  --format table
# exit-code 1 = fail if HIGH or CRITICAL vulnerabilities found

# Also scan for hardcoded secrets (passwords, API keys, tokens)
trivy fs . \
  --scanners secret \
  --exit-code 1

# If clean: "No problems detected"
# If issues found: Trivy shows CVE IDs, severity, and fix versions
```

16.3 Add Trivy to the CI Pipeline

Add these two steps to azure-pipelines.yml BEFORE the packaging step:

Add these two Trivy steps to azure-pipelines.yml before the packaging step — the build fails immediately if a HIGH/CRITICAL CVE or hardcoded secret is detected.

```
# ──────────────────────────────────────────────────────────────
# SECURITY STEP A — FILE SYSTEM VULNERABILITY SCAN
# ──────────────────────────────────────────────────────────────
- script: |
    echo "=== Security Scan: File System ==="
    trivy fs . \
      --severity HIGH,CRITICAL \
      --exit-code 1 \
      --format sarif \
      --output trivy-fs-report.sarif \
      --quiet
    echo "File system scan: PASSED"
  displayName: "Trivy — File System Vulnerability Scan"

# ──────────────────────────────────────────────────────────────
# SECURITY STEP B — HARDCODED SECRETS DETECTION
# ──────────────────────────────────────────────────────────────
- script: |
    echo "=== Security Scan: Secrets Detection ==="
    trivy fs . \
      --scanners secret \
      --exit-code 1 \
      --format table
    echo "Secrets scan: PASSED"
  displayName: "Trivy — Secrets Detection Scan"

# Publish SARIF report as a build artifact
- task: PublishBuildArtifacts@1
  inputs:
    pathToPublish: "trivy-fs-report.sarif"
    artifactName: "security-reports"
  displayName: "Publish Security Scan Report"
```

```
  condition: always()  # Publish even if scan failed
```

Commit the Trivy security scan additions to the CI pipeline and push to a feature branch.

```
git checkout -b feature/add-trivy-security-scan
git add azure-pipelines.yml
git commit -m "feat: Add Trivy security scanning to CI pipeline AB#24"
git push origin feature/add-trivy-security-scan
# PR → merge → Story #24 resolves
```

Part 17 — Azure Key Vault Secret Management

What we'll do: Create an Azure Key Vault, store secrets there, and link it to the pipeline via a variable group. Secrets are retrieved at runtime — never stored in YAML or Git.

🎯 Security Problem

"Our pipeline has the database connection string hardcoded in azure-pipelines.yml.
Everyone who can read the repo can see the password. This violates our security policy."

Solution: Store all secrets in Azure Key Vault. The pipeline retrieves them at runtime — never stored in Git.

17.1 Create the Key Vault and Add Secrets

Create an Azure Key Vault, add the database connection string and API key as secrets, then grant the pipeline service principal read access.

```
# Create Key Vault
az keyvault create \
  --name kv-Ahmed-App-devops \
  --resource-group rg-Ahmed-App-devops \
  --location canadacentral \
  --sku standard

# Add secrets to the vault
az keyvault secret set \
  --vault-name kv-Ahmed-App-devops \
  --name "portal-db-connection" \
  --value "Server=ro10;Database=Ahmed-App;User=admin;Password=S3cur3P@ss!"

az keyvault secret set \
  --vault-name kv-Ahmed-App-devops \
  --name "portal-api-key" \
  --value "sk-Ahmed-App-abc123xyz789"

# Grant the Service Principal read access to secrets
SP_ID=$(az ad sp list --display-name "azure-Ahmed-App-portal" --query "[0].appId" -o tsv)

az keyvault set-policy \
  --name kv-Ahmed-App-devops \
  --spn $SP_ID \
  --secret-permissions get list
```

17.2 Link Key Vault to the Pipeline

57. In Azure DevOps → Pipelines → Library → Variable groups.
58. Click "+ Variable group" and configure:

Name	Ahmed-App-keyvault-secrets
Link secrets from Azure Key Vault	Toggle ON
Azure subscription	azure-Ahmed-App-portal (your service connection)
Key vault name	kv-Ahmed-App-devops
Variables to add	portal-db-connection and portal-api-key

59. Click Save.

Add the variable group to azure-pipelines.yml:

Reference the variable group in azure-pipelines.yml so the pipeline reads Key Vault secrets at runtime. Secrets are masked in all log output.

```
# At the top of azure-pipelines.yml, under "variables:"
variables:
  - group: Ahmed-App-keyvault-secrets  # Links Key Vault secrets as variables
  - name: portalPath
    value: "/var/www/html/portal"

# Use the secret in a pipeline step (it is masked in logs automatically):
- script: |
    echo "Verifying Key Vault secrets are available..."
    if [ -z "$(portal-db-connection)" ]; then
      echo "ERROR: portal-db-connection secret is empty!"
      exit 1
    fi
    echo "Secrets loaded from Key Vault successfully."
  displayName: "Verify Key Vault Secrets Available"
  env:
    DB_CONNECTION: $(portal-db-connection)  # Mapped to env var (masked in logs)
```

Key Vault Best Practices

- Never store secrets in YAML files, environment variables in code, or Git repositories
- Key Vault integration: pipeline retrieves secrets at runtime — never persisted in logs
- Secret masking: Azure DevOps automatically masks any variable group secret in log output
- Rotation: rotate secrets in Key Vault without changing the pipeline — zero downtime
- RBAC: only the service principal needs Get/List permissions on secrets

Commit the Key Vault variable group integration to the CI pipeline.

```
git checkout -b feature/add-key-vault-integration
git add azure-pipelines.yml
git commit -m "feat: Integrate Azure Key Vault secret management in pipeline AB#25"
git push origin feature/add-key-vault-integration
# PR → merge → Story #25 resolves
```

Part 18 — Azure Test Plans

What we'll do: Create a formal test plan in Azure Test Plans with test suites and test cases. Execute the tests manually and link results to work items. Automated smoke tests from the CI pipeline feed into the same test plan.

18.1 Create a Test Plan

60. Go to Test Plans → "+ New test plan".
61. Name: Ahmed-App Portal — Sprint 1 Acceptance Tests.
62. Area path: Ahmed-App-Linux | Iteration: Sprint 1 → Create.

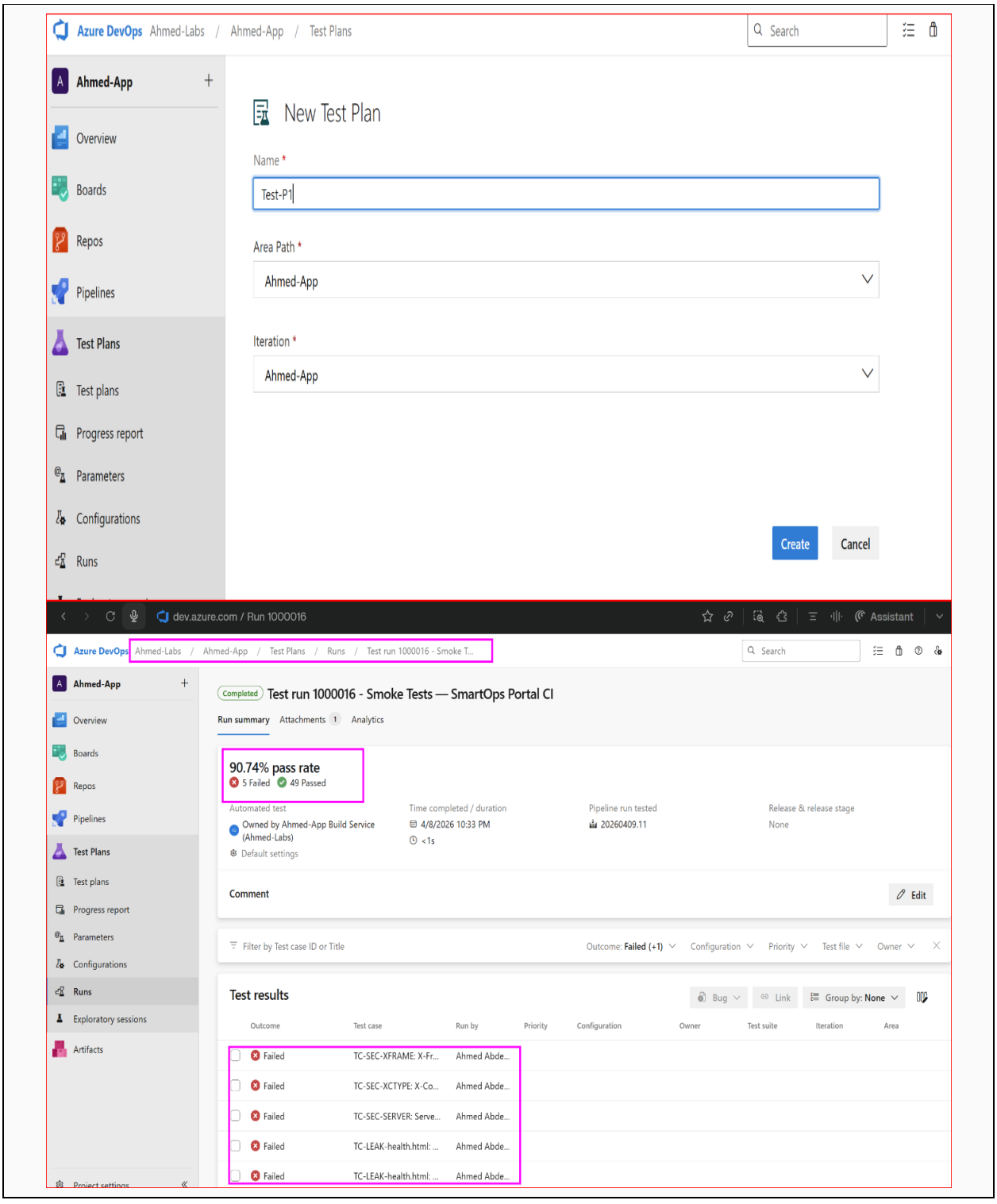
18.2 Create Test Suites and Test Cases

63. In the test plan, click "+ New suite" → Static suite → Name: Navigation Tests.
64. Click "+ New Test Case" inside the suite and add the following cases:

Test Case	Title	Steps → Expected Result
TC-01	Verify portal homepage loads	Open http://ro10:8967/ → Page shows "Ahmed-App IT Operations Portal"
TC-02	Verify health page navigation	Click Health link → health.html loads with HTTP 200
TC-03	Verify SSH log page navigation	Click SSH link → ssh.html loads with HTTP 200
TC-04	Verify all 5 nav links work	Click each link → all pages return HTTP 200
TC-05	Verify staging matches production	Open port 8968 → same content as port 8967

18.3 Execute the Tests and Link to Work Items

65. Click "Run" on the test plan → "Run for web application".
66. Mark TC-01 through TC-04 as Passed.
67. For TC-05: check staging — if it matches, mark Passed; else mark Failed.
68. When marking TC-05 Failed: click "Create bug" → Title: "Staging does not match production" → link to Story #15.
69. Link test cases to User Stories: open each test case → "Links" tab → link to the relevant story.
70. Mark Work Item #26 as Resolved.



Azure Test Plans

Part 19 — Application Insights Monitoring

> **What we'll do:** Create an Application Insights resource, add the JavaScript telemetry snippet to the portal, configure availability tests, and watch live traffic in the Live Metrics stream.

19.1 Create Application Insights Resource

Create a Log Analytics workspace and Application Insights component in Azure, then store the connection string securely in Key Vault.

```
# Create Log Analytics workspace (required for modern App Insights)
az monitor log-analytics workspace create \
  --resource-group rg-Ahmed-App-devops \
  --workspace-name law-Ahmed-App \
  --location canadacentral

WORKSPACE_ID=$(az monitor log-analytics workspace show \
  --resource-group rg-Ahmed-App-devops \
  --workspace-name law-Ahmed-App \
  --query id -o tsv)

# Create Application Insights component
az monitor app-insights component create \
  --app ai-Ahmed-App-portal \
  --location canadacentral \
  --resource-group rg-Ahmed-App-devops \
  --workspace $WORKSPACE_ID \
  --kind web \
  --application-type web

# Get the connection string (use this in your app)
CONNECTION_STR=$(az monitor app-insights component show \
  --app ai-Ahmed-App-portal \
  --resource-group rg-Ahmed-App-devops \
  --query connectionString -o tsv)

# Store the connection string securely in Key Vault
az keyvault secret set \
  --vault-name kv-Ahmed-App-devops \
  --name "appinsights-connection" \
  --value "$CONNECTION_STR"
```

19.2 Add Monitoring to the Portal

Add the Application Insights JavaScript snippet to index.html and push to a feature branch.

```
cd /var/www/html/portal
git checkout -b feature/add-app-insights

# Add the App Insights JavaScript snippet before </head> in index.html
# Get the full snippet from Azure Portal → App Insights → "Getting Started"
# Insert the snippet into index.html with your actual connection string

# The last line of the snippet should call:
# appInsights.trackPageView();

git add index.html
git commit -m "feat: Add Application Insights monitoring to portal AB#27"
git push origin feature/add-app-insights
# PR → merge → Story #27 resolves
```

19.3 Create Alerts for Portal Availability

71. Go to Azure Portal → Application Insights → ai-Ahmed-App-portal.
72. Click "Availability" → "+ Add standard test".
73. Configure:

Test name	Ahmed-App Portal Health Check
URL	https://Ahmed-App-portal-prod.azurewebsites.net/
Test frequency	Every 5 minutes
Test locations	Select 3 regions (East US, West Europe, Southeast Asia)
Success criteria	HTTP status 200
Alert rule	Email when availability < 80%

74. Save → within 5 minutes you will see green heartbeats in the Availability chart.

19.4 View Live Telemetry

75. In Application Insights → "Live Metrics" — open in one tab.
76. In another tab, browse to https://Ahmed-App-portal-prod.azurewebsites.net/
77. Watch the request count spike in Live Metrics in real time!
78. Go to "Failures" — check for any 404 or 500 errors.
79. Go to "Performance" — see page load times across all portal pages.