Pipeline Usage Guide

📋 Overview

This guide explains how to use the various Azure DevOps pipelines included in this solution.

Pipeline Types

1. Multi-Environment Pipeline

File: `azure-pipelines-enhanced.yml`

Purpose: Deploy infrastructure across multiple environments (dev → test → UAT → prod)

When to Use

- Regular releases
- Full environment deployments
- Progressive rollout across environments

How to Run

1. Navigate to **Pipelines** in Azure DevOps
2. Select **Bicep Infrastructure - Multi-Environment**
3. Click **Run pipeline**
4. Configure parameters:

```
Branch: main

Parameters:
  applicationName: step
  environmentsToDeploy:
    - dev
    - test
    - uat
    - prod
  skipValidation: false
  skipWhatIf: false
  deploymentMode: Standard
```

5. Click **Run**

Pipeline Stages

```
1. Build & Lint
   ├── Lint Bicep files
   ├── Build templates to ARM
   └── Publish artifacts

2. Validate
   ├── Validate Dev template
   ├── Validate Test template
   ├── Validate UAT template
   └── Validate Prod template

3. Preview (What-If)
   ├── Preview Dev changes
   ├── Preview Test changes
   ├── Preview UAT changes
   └── Preview Prod changes

4. Deploy Dev
   └── Deploy to Development

5. Deploy Test
   └── Deploy to Test (after Dev succeeds)

6. Deploy UAT
   ├── Wait for manual approval ⏸
   └── Deploy to UAT

7. Deploy Prod
   ├── Wait for manual approval ⏸
   ├── Final approval gate ⏸
   ├── Deploy to Production
   └── Post-deployment verification
```

Environment-Specific Behavior

Environment	Auto-Deploy	Approval Required	What-If	Verification
Dev	✅	❌	Optional	✅
Test	✅	❌	✅	✅
UAT	❌	✅	✅	✅
Prod	❌	✅ (Double)	✅	✅ Enhanced

---

2. Single Environment Pipeline

File: `pipelines/examples/single-environment-pipeline.yml`

Purpose: Deploy to a single environment only (typically dev)

When to Use

- Rapid development cycles
- Testing template changes
- Feature branch deployments
- Development iterations

How to Run

1. Create a new pipeline from `pipelines/examples/single-environment-pipeline.yml`
2. Triggered automatically on:
   - Commits to `develop` branch
   - Commits to `feature/*` branches
3. Or run manually

Quick Deployment to Dev

```
# Commit and push to develop branch
git checkout develop
git add applications/step/
git commit -m "Update VM configuration"
git push

# Pipeline auto-triggers and deploys to dev
```

---

3. Hotfix Pipeline

File: `pipelines/examples/hotfix-pipeline.yml`

Purpose: Emergency deployments that skip normal process

⚠ **WARNING:** Use only for genuine emergencies!

When to Use

- Critical production issues
- Security patches
- Emergency fixes
- Incidents requiring immediate resolution

How to Run

1. Navigate to **Hotfix Deployment** pipeline
2. Click **Run pipeline**

3. **Required parameters:**

```
Branch: main (or hotfix branch)

Parameters:
  targetEnvironment: prod
  applicationName: step
  justification: "Critical security patch for CVE-XXXX"
  changeTicketNumber: "CHG-12345"
```

4. Click **Run**
5. **Approval required** at validation stage
6. Review what-if analysis
7. **Final approval** before deployment
8. Monitor deployment closely
9. Perform manual verification

Hotfix Process

```
1. Hotfix Validation
   ├── Display hotfix information
   └── Manual approval required ⏸

2. Hotfix Deploy
   ├── Quick template validation
   ├── What-If analysis (required)
   ├── Execute deployment
   ├── Tag resources as hotfix
   └── Post-hotfix verification

3. Post-Deployment
   └── Manual smoke tests required ⚠
```

Post-Hotfix Checklist

- ☐ Verify deployment succeeded
- ☐ Test affected functionality
- ☐ Monitor for issues (15-30 minutes)
- ☐ Update change ticket
- ☐ Document what was deployed
- ☐ Plan follow-up normal deployment
- ☐ Update team on resolution

---

4. Multi-Application Pipeline

File: `pipelines/examples/multi-application-pipeline.yml`

Purpose: Deploy multiple applications in parallel or sequentially

When to Use

- Deploying multiple applications
- Coordinated releases
- Infrastructure-wide updates

How to Run

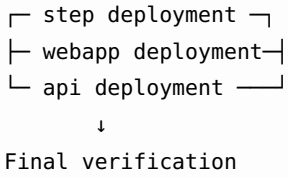
```yaml
Parameters:
  applications:
    - step
    - webapp
    - api
  deploymentStrategy: Sequential  # or Parallel
  targetEnvironment: dev
```

Deployment Strategies

Sequential:

```
step deployment
  ↓ (success)
webapp deployment
  ↓ (success)
api deployment
  ↓ (success)
Final verification
```

Parallel:



When to Use Each Strategy

Strategy	Best For	Pros	Cons
Sequential	Dependencies between apps	Controlled, ordered	Slower
Parallel	Independent apps	Fast, efficient	Resource intensive

---

5. PR Validation Pipeline

File: `pipelines/examples/pr-validation-pipeline.yml`

Purpose: Validate Bicep templates in pull requests

When to Use

- Automatically triggered on PRs to main/develop
- Ensures code quality before merge

What It Does

```
1. Lint Bicep Files
   └── Lint only changed .bicep files

2. Validate Templates
   ├── Validate for Dev
   ├── Validate for Test
   ├── Validate for UAT
   └── Validate for Prod

3. What-If Analysis
   └── Show preview of changes

4. Publish Results
   └── Comment on PR with results
```

PR Validation Results

You'll see a comment on your PR:

```
## 🔍 Bicep Validation Results

**PR:** #42
**Build:** [Build 20250108.1](link)

### Results Summary

| Stage | Result |
|-------|--------|
| Linting | ✓ Passed |
| Validation | ✓ Passed |
| What-If | ✓ Passed |

### Next Steps

- Review the what-if analysis to understand the changes
- Ensure all tests pass before merging
- Get required approvals
```

Common Scenarios

Scenario 1: Deploy New Feature to Dev

```
# Create feature branch
git checkout -b feature/add-monitoring

# Make changes
vim applications/step/main.bicep

# Commit and push
git add .
git commit -m "Add monitoring configuration"
git push -u origin feature/add-monitoring

# Create PR - PR validation runs automatically
# Merge PR after approval
# Single environment pipeline deploys to dev
```

Scenario 2: Promote to Production

```
# Ensure changes are in main branch
git checkout main
git pull

# Run multi-environment pipeline
# Select environments: dev, test, uat, prod
# Approve UAT deployment when ready
# Approve Production deployment when ready
```

Scenario 3: Rollback Production

```
# Option 1: Redeploy previous version
# Run pipeline with previous commit
Branch: main
Commit: <previous-good-commit>

# Option 2: Use rollback deployment
# Manually restore from backup
az deployment sub create \
  --name "rollback-$(date +%s)" \
  --location eastus \
  --template-file backup-template.json
```

Scenario 4: Update Single Environment

```
# Run multi-environment pipeline
```

```yaml
Parameters:
  environmentsToDeploy:
    - test  # Only update test
  skipValidation: false
  skipWhatIf: false
```

Scenario 5: Change VM Size

1. Update parameter file:

   ```
   vim applications/step/prod/prod.bicepparam
   # Change vmSize parameter
   ```

2. Run what-if to see impact:

   ```
   az deployment sub what-if \
     --location eastus \
     --template-file applications/step/main.bicep \
     --parameters applications/step/prod/prod.bicepparam
   ```

3. Run pipeline to deploy

4. Monitor VM during resize

Pipeline Parameters

Available Parameters

applicationName

```yaml
Type: string
Default: 'step'
Options: [step, webapp, api, all]
Description: Which application to deploy
```

environmentsToDeploy

```yaml
Type: array
Default: [dev, test, uat, prod]
Options: Any combination of [dev, test, uat, prod]
Description: Which environments to deploy to
Example: [dev, test]  # Deploy to dev and test only
```

skipValidation

```yaml
Type: boolean
Default: false
Description: Skip template validation stage
```

```
Use When: Validation already done, rapid iteration
```

skipWhatIf

```
Type: boolean
Default: false
Description: Skip what-if analysis
Use When: No changes expected, redeployment
```

deploymentMode

```
Type: string
Default: 'Standard'
Options: [Standard, HotFix, Rollback]
Description: Type of deployment
```

Monitoring Pipeline Runs

View Pipeline Status

1. **Navigate to Pipelines**
2. **View recent runs:**
   - Green checkmark: Success
   - Red X: Failed
   - Orange circle: In progress
   - Gray pause: Waiting for approval

View Logs

1. Click on pipeline run
2. Click on stage (e.g., "Deploy Dev")
3. Click on job (e.g., "Deploy Infrastructure")
4. Click on task to see detailed logs

Download Artifacts

1. Click on pipeline run
2. Click **Artifacts** tab
3. Available artifacts:
   - `arm-templates`: Built ARM templates
   - `bicep-source`: Source code
   - `deployment-logs-*`: Deployment logs per environment
   - `whatif-results-*`: What-if analysis results

Troubleshooting Pipeline Runs

Pipeline Stuck on Approval

Issue: Pipeline waiting for approval

Solution: 1. Check email for approval notification 2. Or go to **Pipelines > Environments >** [Environment] 3. Click **View history** 4. Click **Approve** or **Reject**

Stage Failed

Issue: Stage shows red X

Steps: 1. Click on failed stage 2. Review error messages 3. Common issues: - Validation errors: Fix template - Permission errors: Check service connection - Resource conflicts: Check Azure Portal 4. Fix issue 5. Re-run pipeline

Re-run Pipeline

```
Option 1: Re-run entire pipeline
  - Click "Run new"

Option 2: Re-run failed stage
  - Click "Rerun failed jobs"
  - Only failed stage runs

Option 3: Re-run from specific stage
  - Not available (use "Run new")
```

Best Practices

1. Always Review What-If

✕ Don't:

```yaml
skipWhatIf: true  # Skip what-if
```

✅ **Do:**

```yaml
skipWhatIf: false  # Always review
# Review what-if results before approving
```

2. Progressive Deployment

✕ Don't:

```yaml
environmentsToDeploy: [prod]  # Deploy directly to prod
```

✅ **Do:**

```
Run 1: [dev]
Run 2: [dev, test]
Run 3: [dev, test, uat]
Run 4: [dev, test, uat, prod]
```

3. Use Appropriate Pipeline

Scenario	Pipeline	Reason
Regular release	Multi-environment	Full validation
Quick dev test	Single environment	Fast iteration
Emergency	Hotfix	Skip some stages
PR review	PR validation	Pre-merge checks

4. Monitor Costs

Pipeline runs consume: - Build minutes - Agent time - Storage for artifacts

Optimize by: - Cleaning up old runs - Removing unused artifacts - Using self-hosted agents for heavy workloads

---

Pipeline Maintenance

Regular Tasks

Weekly: - Review failed pipeline runs - Clean up old artifacts - Update documentation if process changed

Monthly: - Review and rotate secrets - Update pipeline templates if needed - Check for Azure DevOps updates - Review pipeline efficiency

Quarterly: - Service principal permission review - Update approval group memberships - Review deployment patterns - Optimize pipeline performance

Updating Pipelines

```
# Update pipeline YAML
git checkout -b update/pipeline-improvements
vim azure-pipelines-enhanced.yml

# Test in dev first
git add azure-pipelines-enhanced.yml
git commit -m "Update pipeline template"
```

```
git push

# Create PR and review
# Merge after validation
```

Additional Resources

- Azure DevOps Setup Guide
- Quick Reference
- Troubleshooting Guide

Document Version: 1.0
Last Updated: January 2025