**Wiz Security Tool - Implementation Guide for Unity**

**1. How Wiz Will Be Consumed in Unity**

**Primary Method: CI/CD Pipeline (GitHub Actions)**

**Automatic scanning on every Pull Request to main/release branches**

Triggered automatically when:

- Developer creates PR to main/release

- Terraform/Bicep/ARM files modified

- Scans IaC code before deployment

**Benefits:**

* Prevents security issues before deployment
* Blocks merge if critical vulnerabilities found
* No manual intervention needed

**Secondary Method: Wiz Portal**

**URL:** https://app.wiz.io

**Used for:**

* Viewing security dashboard
* Monitoring Azure resources in real-time
* Generating compliance reports
* Managing policies

**Access:** Security team and DevOps leads (read-only for most users)

**Tertiary Method: Wiz CLI**

**Used for:** Local development testing (optional)

curl -o wizcli https://wizcli.app.wiz.io/latest/wizcli

chmod +x wizcli

sudo mv wizcli /usr/local/bin/

# Scan locally

wizcli iac scan --path ./terraform --output results.json

**2. Who Manages the Tool**

**Wiz Administrator**

* **Role:** Platform/Security Admin
* **Responsibilities:** User management, policy configuration, integrations, license monitoring
* **Access:** Full admin access to Wiz portal

**Platform Security Officer (PSO)**

* **Role:** Security Lead
* **Responsibilities:** Approve exceptions, review high-priority vulnerabilities, compliance oversight
* **Access:** Security configuration access

**DevOps Team Leads**

* **Role:** Pipeline Maintainers
* **Responsibilities:** Maintain GitHub Actions workflows, troubleshoot scan failures, train developers
* **Access:** Repository admin, can manage service accounts

**Developers**

* **Role:** End Users
* **Responsibilities:** Fix identified vulnerabilities, review scan results in PRs
* **Access:** View results via GitHub PR comments only

**3. Main vs Release Branch Management**

**Branch Strategy**

| **Branch** | **Scan Policy** | **Blocking Criteria** | **Approvals Required** |
| --- | --- | --- | --- |
| **Feature** | Informational | No blocking | None |
| **Main** | Strict | Block on Critical | 1 reviewer |
| **Release** | Very Strict | Block on Critical + High | 2 reviewers + PSO |

**Main Branch Rules**

**GitHub Settings:**

Repository → Settings → Branches → Add protection rule

- Branch: main

- Require PR before merge: ✓

- Require status check: "Wiz IaC Security Scan" ✓

- Require 1 approval: ✓

**Wiz Policy:**

* **BLOCK:** Any Critical severity issue
* **WARN:** High severity issues (>5)
* **REPORT:** Medium/Low issues

**Report Distribution:**

* PR comment with scan results
* Slack notification to #devops
* Artifacts stored for 30 days

**Release Branch Rules**

**GitHub Settings:**

Repository → Settings → Branches → Add protection rule

- Branch: release

- Require PR before merge: ✓

- Require status checks: "Wiz IaC Security Scan" + "Compliance Check" ✓

- Require 2 approvals (1 from Security team): ✓

**Wiz Policy:**

* **BLOCK:** Any Critical OR High severity issue
* **WARN:** Medium severity issues (>10)
* **REPORT:** Low issues
* **REQUIRED:** PSO approval for all merges

**Report Distribution:**

* PR comment with detailed compliance report
* Email to Security team + PSO + Management
* PDF report generated
* Artifacts stored for 90 days (compliance)

**Reporting Differences**

**Main Branch Report:**

* Summary table (Critical/High/Medium/Low counts)
* List of High/Critical findings with file locations
* Remediation recommendations
* Stored 30 days

**Release Branch Report:**

* Everything in Main report PLUS:
* Compliance status (CIS, NIST, ISO)
* PSO approval checklist
* Deployment window details
* Change management ticket reference
* Stored 90 days

**4. Runtime Security of Infrastructure**

**How It Works**

Wiz continuously monitors Azure subscription using **agentless scanning** (no agents on VMs).

**Scanning Frequency:**

* Configuration changes: Every 1 hour
* Vulnerabilities: Every 4 hours
* Network exposure: Continuous
* Compliance: Every 6 hours

**What Wiz Monitors**

**Compute:**

* VMs: OS vulnerabilities, open ports, patch status
* AKS: Container vulnerabilities, Kubernetes misconfigurations
* Container Instances: Image vulnerabilities, network exposure

**Data:**

* Storage Accounts: Public access, encryption status
* SQL Databases: Firewall rules, TDE status
* Cosmos DB: Network restrictions, encryption

**Identity:**

* Service Principals: Overprivileged roles
* Managed Identities: Permission scope
* Role Assignments: Excessive permissions

**Network:**

* NSGs: Inbound/outbound rules from internet
* VNets: DDoS protection, network segmentation
* Exposed Resources: Internet-facing services

**Alert Routing**

| **Severity** | **Destination** | **Response Time** | **Example** |
| --- | --- | --- | --- |
| **Critical** | PagerDuty + Slack #security-critical | 15 minutes | Storage account publicly exposed |
| **High** | Slack #security-alerts + Email | 1 hour | Unpatched critical CVE |
| **Medium** | Email digest (daily) | 1 business day | Missing encryption |
| **Low** | Weekly email digest | Next sprint | Best practice recommendation |

**Integration Setup**

**Slack:**

Wiz Portal → Settings → Integrations → Slack

- Webhook URL: https://hooks.slack.com/services/YOUR/WEBHOOK

- Channels: #security-critical, #security-alerts

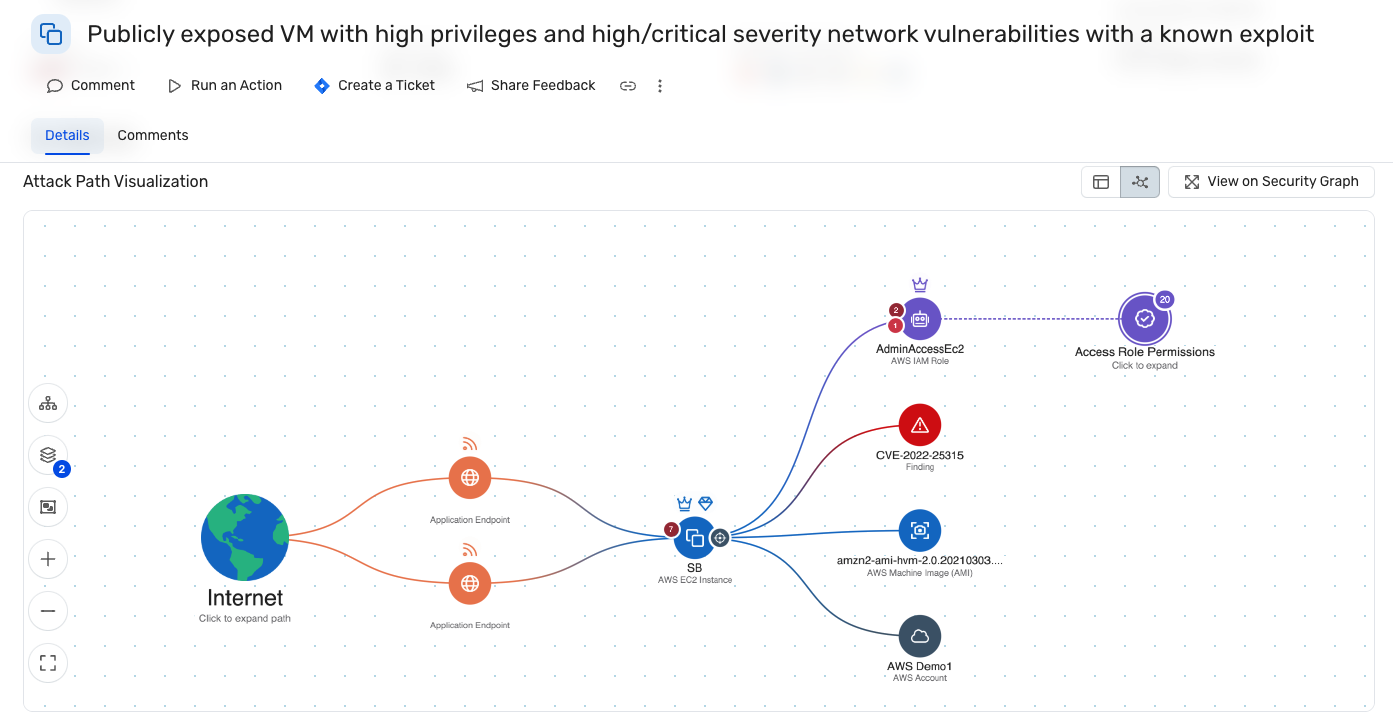
**JIRA:**

Wiz Portal → Settings → Integrations → JIRA

- JIRA URL: https://unity.atlassian.net

- Project: SEC

- Auto-create tickets for Critical/High severity



**5. Vulnerability Prioritization & PSO Approval Process**

**Priority Matrix**

| **Wiz Severity** | **Environment** | **Internet-Facing** | **Priority** | **SLA** |
| --- | --- | --- | --- | --- |
| Critical | Production | Yes | P0 | 1 hour |
| Critical | Production | No | P1 | 24 hours |
| High | Production | Yes | P1 | 24 hours |
| High | Production | No | P2 | 7 days |
| Medium | Production | Any | P3 | 30 days |
| Low | Any | Any | P4 | 90 days |

**Workflow**

1. Wiz Detects Vulnerability

↓

2. Auto-Create JIRA Ticket (for Critical/High)

↓

3. Security Team Triages (4 hours)

↓

4. Assign to Resource Owner

↓

5. Owner Creates Remediation Plan

↓

6. PSO Reviews & Approves (for P0/P1)

↓

7. Implement Fix

↓

8. Wiz Verifies Resolution

↓

9. Close Ticket

**PSO Approval Required For:**

* All P0 (Critical) vulnerabilities
* All P1 (High) in production
* Exception/risk acceptance requests
* Extended timeline requests

**PSO Approval Process**

**Remediation Plan Must Include:**

1. Technical fix description
2. Implementation timeline
3. Testing approach
4. Rollback plan
5. Risk assessment
6. Resource requirements

**PSO Reviews:**

* Adequacy of remediation
* Risk vs. effort trade-off
* Compliance impact
* Alternative approaches

**PSO Decision:**

* Approve as proposed
* Approve with conditions
* Request changes
* Reject - requires revision

**6. Technical Implementation**

**Step 1: Create Wiz Service Account**

1. Login to Wiz Portal as Admin

2. Settings → Service Accounts → Create

3. Name: unity-github-actions

4. Scope: iac:scan

5. Copy Client ID and Client Secret (shown once only)

**Step 2: Configure GitHub Secrets**

1. Repository → Settings → Secrets and variables → Actions

2. Add secrets:

- WIZ\_CLIENT\_ID: [paste Client ID]

- WIZ\_CLIENT\_SECRET: [paste Client Secret]

**Step 3: Create GitHub Actions Workflow**

**File:** .github/workflows/wiz-iac-scan.yml

name: Wiz IaC Security Scan

on:

pull\_request:

branches: [main, release]

paths: ['\*\*.tf', '\*\*.bicep', '\*\*.json']

push:

branches: [main, release]

jobs:

wiz-scan:

runs-on: ubuntu-latest

permissions:

contents: read

pull-requests: write

steps:

- uses: actions/checkout@v4

- name: Download Wiz CLI

run: |

curl -o wizcli https://wizcli.app.wiz.io/latest/wizcli

chmod +x wizcli

sudo mv wizcli /usr/local/bin/

- name: Authenticate to Wiz

env:

WIZ\_CLIENT\_ID: ${{ secrets.WIZ\_CLIENT\_ID }}

WIZ\_CLIENT\_SECRET: ${{ secrets.WIZ\_CLIENT\_SECRET }}

run: |

wizcli auth --id "$WIZ\_CLIENT\_ID" --secret "$WIZ\_CLIENT\_SECRET"

- name: Run Wiz Scan

run: |

wizcli iac scan --path . --output wiz-results.json --output-format json

- name: Check Results

run: |

CRITICAL=$(cat wiz-results.json | jq '.summary.critical // 0')

HIGH=$(cat wiz-results.json | jq '.summary.high // 0')

# Block on critical for main branch

if [[ "$GITHUB\_REF" == "refs/heads/main" && "$CRITICAL" -gt 0 ]]; then

echo "❌ Critical issues found - blocking merge"

exit 1

fi

# Block on critical OR high for release branch

if [[ "$GITHUB\_REF" == "refs/heads/release" && ("$CRITICAL" -gt 0 || "$HIGH" -gt 0) ]]; then

echo "❌ Critical/High issues found - blocking merge"

exit 1

fi

- name: Upload Results

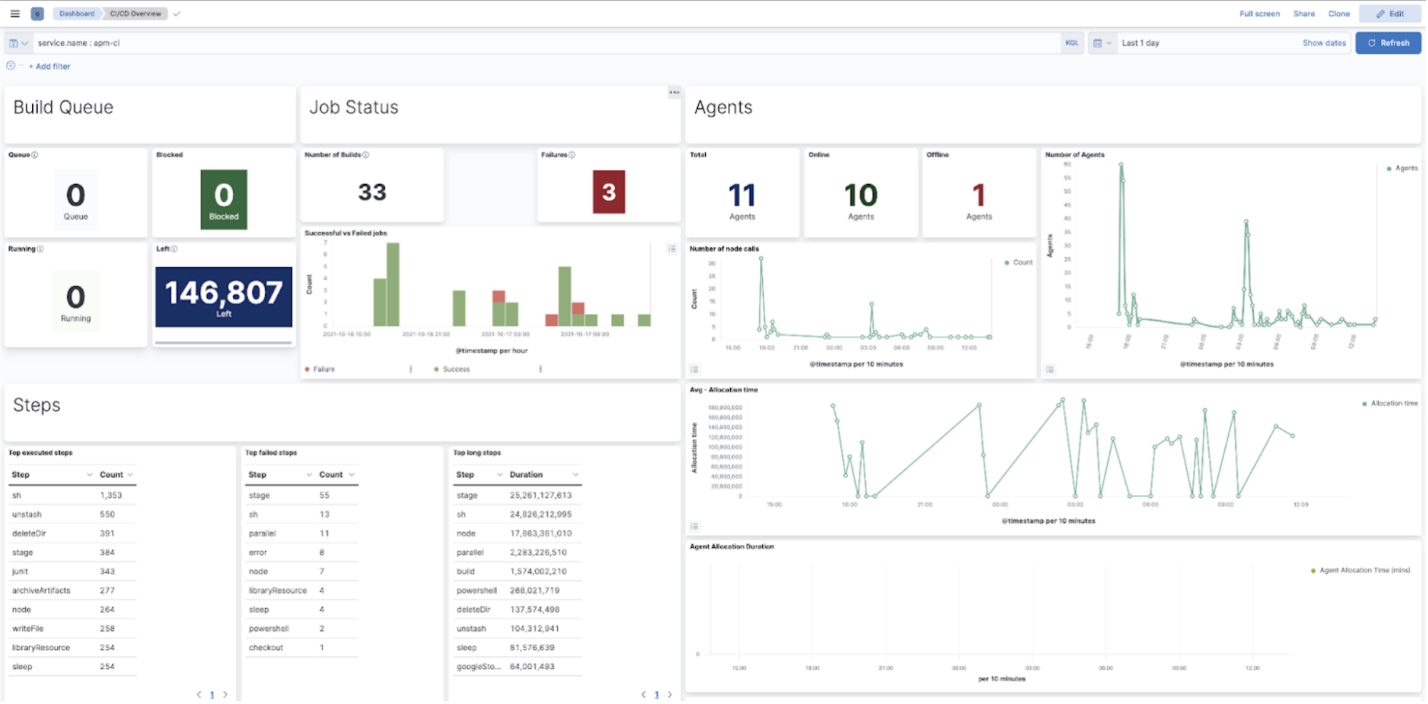
if: always()

uses: actions/upload-artifact@v4

with:

name: wiz-scan-results

path: wiz-results.json



**Step 4: Configure Branch Protection**

Repository → Settings → Branches

For 'main':

- Require pull request: ✓

- Require status check "Wiz IaC Security Scan": ✓

- Require 1 approval: ✓

For 'release':

- Require pull request: ✓

- Require status check "Wiz IaC Security Scan": ✓

- Require 2 approvals: ✓

**Step 5: Test Integration**

# Create test branch

git checkout -b test/wiz-integration

# Add workflow file

git add .github/workflows/wiz-iac-scan.yml

git commit -m "Add Wiz scanning"

git push origin test/wiz-integration

# Create PR and monitor workflow

gh pr create --title "Add Wiz Security" --base main

**7. Common CLI Commands**

**Authentication**

# Authenticate

wizcli auth --id "CLIENT\_ID" --secret "CLIENT\_SECRET"

# Check status

wizcli auth status

**Scanning**

# Basic scan

wizcli iac scan --path .

# Scan with output

wizcli iac scan --path ./terraform --output results.json

# Scan specific policy

wizcli iac scan --path . --policy "Default IaC Policy"

**Analyzing Results**

# View summary

cat results.json | jq '.summary'

# List critical issues

cat results.json | jq '.findings[] | select(.severity=="CRITICAL")'

# Count by severity

cat results.json | jq '.findings | group\_by(.severity) | map({severity: .[0].severity, count: length})'