AZ-400 DevOps Engineer – Visual Notes

Part 1: Exam Roadmap & Study Map

vighnesh-notes

vighneshnaik.com

linkedin.com/in/vighneshnaik6

WHAT IS AZ-400?

The AZ-400 exam validates your ability to **design and implement** Microsoft DevOps solutions.

It's intended for DevOps engineers who combine **people**, **process**, and **technology** to deliver **value** continuously.

ROLE EXPECTATIONS

- Collaborate between development and operations
- Build reliable, resilient systems
- Drive continuous improvement
- Ensure security and compliance
- Automate to deliver faster and safer
- Provide observability and actionable insights

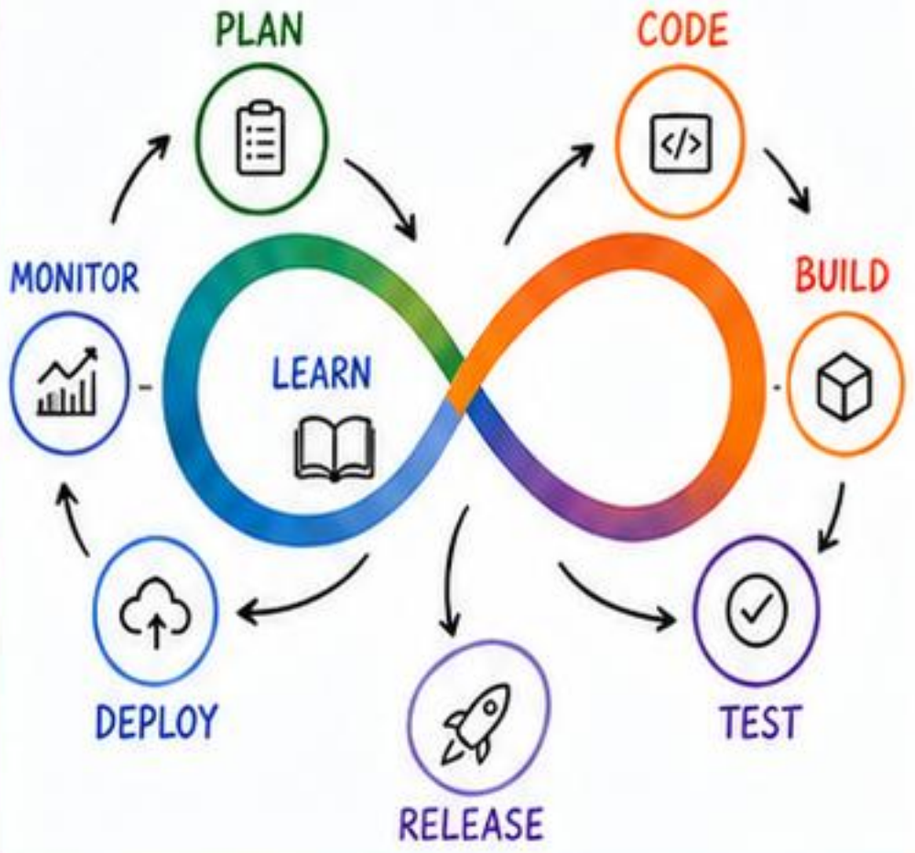
AT-A-GLANCE EXAM FACTS

- **Exam Code:** AZ-400
- **Passing Score:** 700
- **Level:** Intermediate/Advanced
- **Common Question Formats:** Multiple choice, Case study, Drag-and-drop (and more)
- **Duration:** 120 minutes
- **Languages:** English, Japanese, Chinese, Korean, French, German, Spanish

SKILLS OUTLINE & WEIGHTS

1. Processes and communication 10–15%
2. Source control 10–15%
3. Build and release pipelines 50–55%
4. Security and compliance 10–15%
5. Instrumentation strategy 5–10%

THE DEVOPS LOOP



HOW TO USE THIS SERIES

- ✓ This series breaks down the AZ-400 exam domain by domain.
- ✓ Each page focuses on the key topics, concepts, and exam tips you need to know.
- ✓ Pipelines is the largest exam area (~50–55%), so it's split across two pages (4 & 5).

YOUR STUDY STRATEGY

- Spend ~50% of your study time on Build & Release Pipelines.
- Practice YAML, GitHub Actions and Azure Pipelines.
- Understand deployment strategies, environments, approvals & checks.
- Know IaC (Bicep/Terraform), secrets, and configuration management.
- Don't skip monitoring, traceability and feedback loops.

THE DEVOPS MINDSET

- Automate everything possible.
- Secure by design.
- Measure what matters.
- Make small, reversible changes.
- Collaborate and communicate continuously.

WHAT'S NEXT? YOUR STUDY ROADMAP

Page 2 — 1 PROCESSES & COMMUNICATIONS 10–15%

Page 3 — 2 SOURCE CONTROL 10–15%

Page 4 — 3 PIPELINES FOUNDATIONS (Part 1) ~25–30%

Page 5 — 3 PIPELINES DELIVERY & IaC (Part 2) ~25–30%

Page 6 — 4 SECURITY & COMPLIANCE 10–15%

Page 7 — 5 INSTRUMENTATION & OBSERVABILITY 5–10%

Stay consistent, practice hands-on, and think like a DevOps engineer.

Build. Test. Learn. Repeat.

✓ You've got this! 😀

AZ-400 DevOps Engineer – Visual Notes

Part 2: Processes & Communications

vighnesh-notes
vighneshnaik.com
linkedin.com/in/vighneshnaik6

1 WHY THIS DOMAIN MATTERS

Great DevOps starts with clear communication, visible work, and traceable flow.

✓ Well-defined processes and tools align teams, reduce waste, and create predictable outcomes.

✓ This domain ensures everyone knows what to build, why, and how it's progressing.

COVERS

- ☑ Planning and tracking work
- ☑ Collaboration and communication
- ☑ Traceability across the delivery lifecycle
- ☑ Documentation and knowledge sharing
- ☑ Metrics, reporting, and integrations

CORE MINDSET

Make work visible.
Keep teams aligned.
Measure flow.
Continuously communicate.
Improve together.

2 PLAN AND TRACK WORK 10–15%

GitHub Issues
- Capture ideas, tasks, bugs, and epics
- Use labels, milestones, assignees, and templates

GitHub Projects
- Visualize work on boards or tables
- Automate with workflows and custom fields

Azure Boards
- Work items: Epic, Feature, User Story, Bug, Task
- Backlogs, sprints, sprint planning

Backlogs & Boards
- Prioritize work in product backlog
- Track progress on Kanban or Scrum boards

Queries & Dashboards
- Build queries (WIQL, filters) to find work
- Dashboards for team and portfolio visibility

3 COLLABORATION

GitHub Discussions
Discuss ideas, Q&A, announcements

Pull Request Conversations
Inline comments, reviews, approvals

Notifications
Stay informed with mentions, subscriptions, and alerts

Microsoft Teams Integration
Link repos, get PR/build notifications in Teams, chatops workflows

4 END-TO-END TRACEABILITY

Link everything to see the full picture.

Plan	Epics, Features, User Stories, Tasks
Code	Commits, Branches, Pull Requests
Build	CI builds, pipeline runs, artifacts
Test	Automated & manual tests, test results
Release	Releases, approvals, environments
Production	Actual changes, work items, incidents

Traceability + Visibility + Accountability

5 DOCUMENTATION & KNOWLEDGE SHARING

Wikis
Centralize how-tos, architecture, runbooks, and team knowledge

Release Notes
Summarize what changed, why, and impact

Markdown
Simple, portable, works everywhere (GitHub, Azure DevOps)

API Documentation
Document APIs clearly for consumers

Mermaid Diagrams
Create flowcharts, sequence diagrams, architectures

OpenAPI / Swagger
Standardize and share API contracts

Autogenerate docs from code comments and OpenAPI specs to keep documentation accurate and up to date.

METRICS THAT MATTER

Lead Time	Time from work item creation to delivery
Cycle Time	Time from start of work to completion
Deployment Frequency	How often you successfully deploy to production
Change Failure Rate	% of deployments that cause a failure
Mean Time to Restore (MTTR)	Time to restore service after a failure
Dashboard Reporting	Visualize trends, monitor health, drive continuous improvement

7 INTEGRATIONS & AUTOMATION

Webhooks
Send real-time events to external systems

Service Hooks
Trigger actions on events (builds, PRs, work items)

Teams & ChatOps
Automate workflows, notify, and act from chat

Email Notifications
Keep stakeholders informed

Jira / ServiceNow
Integrate ITSM & ALM processes

8 EXAM TIPS / COMMON BEST-ANSWER PATTERNS

- Focus on the outcome. Choose options that improve visibility, alignment, and traceability.
- Favor built-in platform capabilities over custom solutions.
- Automate status updates and notifications.
- Integrations that connect planning → code → build → test → test → release → best!
- Pick the MOST comprehensive answer (end-to-end visibility, automation, reporting).
- Security + compliance: Use role-based access, audit trails, approvals.
- Look for answers that reduce manual work and improve collaboration.
- Communicate early and often—make information easy to find.

REMEMBER

1. Connect planning to delivery. Everything links.
2. Measure flow, not just output.
3. Document continuously. Keep it current.
4. Automate reporting and notifications. Save time.
5. Collaborate early and empower teams.

Stay consistent, practice hands-on, and think like a DevOps engineer.
Build. Test. Learn. Repeat.

AZ-400 DevOps Engineer — Visual Notes

3/7

Part 3: Source Control Strategy

SOURCE CONTROL (10–15%)

Design and implement a source control strategy that supports collaboration, quality, and secure delivery.

vighnesh-notes

vighneshnaik.com

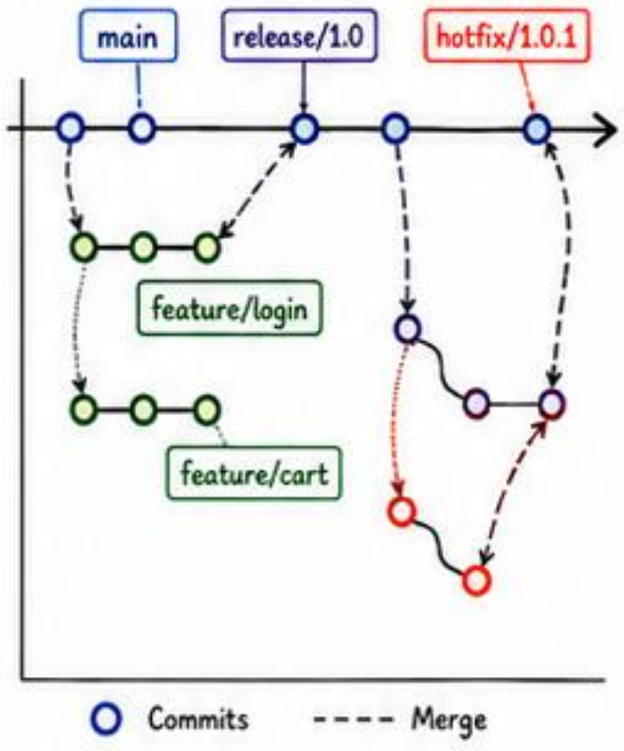
linkedin.com/in/vighneshnaik6

WHAT YOU'LL DO

- ☑ Use Git effectively
- ☑ Protect code and branches
- ☑ Organize repos and scale for teams
- ☑ Recover from issues
- ☑ Keep history clean and safe

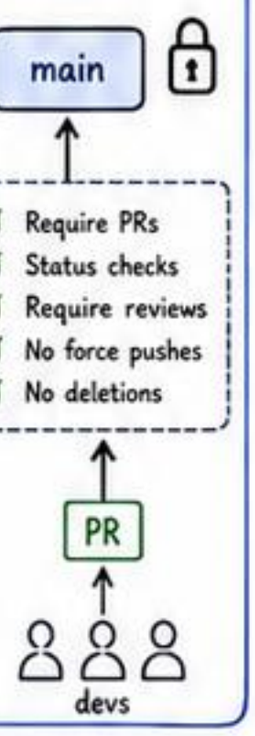
1 BRANCHING STRATEGIES

- **Trunk-based development**
  Short-lived branches, frequent commits, continuous integration.
- **Feature branches**
  Isolate work, open PR early, merge small & often.
- **Release branches**
  Stabilize for release, bug fixes only.
- **Hotfix branches**
  Urgent fixes from main, merge back quickly.



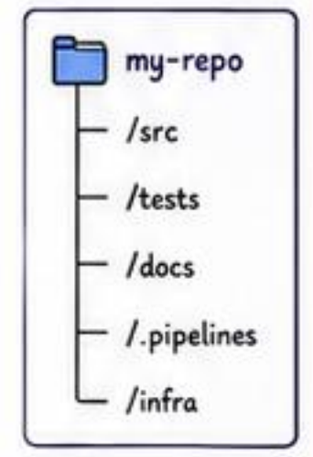
2 PULL REQUEST PROTECTION

- **Required reviewers** — Ensure code is reviewed.
- **Status checks** — All checks must pass.
- **Build validation** — CI build must succeed.
- **Branch policies** — Enforce rules consistently.
- **Branch protection rules** — Lock down important branches.
- **Restrict direct pushes** — Changes only via pull requests.



3 REPOSITORY MANAGEMENT

- **Permissions** — Least privilege access.
- **Tags** — Mark important points (v1.0.0).
- **Releases** — Package and publish deliverables.
- **Access control** — Teams, roles, scopes.
- **Repo organization** — Consistent structure, README, templates.



4 LARGE REPOS AND FILES

- **Git LFS** — Store large files outside Git history.
- **git-fat (legacy)** — Older tool for large file management.
- **Scalar (Microsoft tool)** — Optimize large repos for better performance.

LARGE FILES WITH GIT LFS

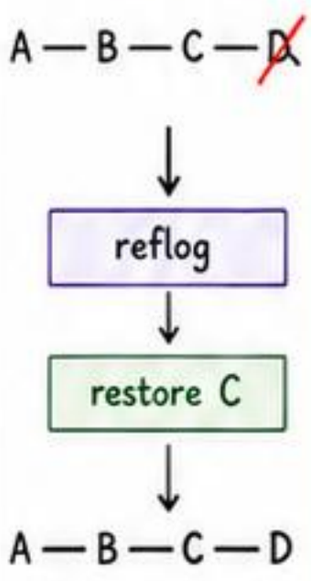
Monorepo vs Multi-repo

Monorepo	Multi-repo
✓ Simpified commits	✓ Clear ownership
✓ Shared changes	✓ Independent scaling
✓ Harder boundaries	✗ More overhead
✓ Slower at scale	✗ Cross-repo changes

5 GIT RECOVERY ESSENTIALS

revert	Create a new commit that undoes changes.
restore	Discard local changes in working directory.
reset	Move HEAD (carefully!). --soft | mixed | hard
reflog	Recover lost commits by viewing HEAD history.
cherry-pick	Apply a specific commit onto current branch.
rebase	Reapply commits on top of another base.
merge	Combine histories, keep both paths.

EXAMPLE: RECOVER A LOST COMMIT



6 SECRET INCIDENTS

- **Remove exposed data from history** — Use BFG or filter-repo.
- **Rotate credentials** — Invalidate and regenerate immediately.
- **Prevent recurrence** — Scan + policy + training.

TOOLS

Gitleaks · TruffleHog · GitHub Secret Scanning

7 COLLABORATION PATTERNS AND CLEAN COMMIT HABITS

- **Small, focused commits** — One change per commit.
- **Clear commit messages** — Explain the what and why.
- **Pull early, push often** — Stay up to date.
- **Open PRs early** — Get feedback fast.
- **Resolve conflicts early** — Communicate and sync.

GOOD COMMIT MESSAGE

```
feat: add login validation
Add email format and password
length validation.
Fixes #123
```

COMMIT HISTORY (CLEAN IS BETTER)

8 EXAM TIPS / CHOOSE-THE-BEST-BRANCH-STRATEGY

Think about the scenario:

- ☑ Team size & experience
- ☑ Deployment frequency
- ☑ Need for traceability
- ☑ Release cadence
- ☑ Compliance needs
- ☑ Repo size & complexity

QUICK GUIDE

Small team, high velocity	Trunk-based
Medium team, features in flight	Feature branches
Stable releases required	Release branches
Urgent production fix	Hotfix branches
Multiple products, big team	Multi-repo
Shared platform, many teams	Monorepo

BEST PRACTICES (RECAP)

- **Protect main** — Branch policies, no direct pushes.
- **Review changes** — Use PRs, reviewers, and status checks.
- **Keep history clean** — Small commits, clear messages, no secrets.
- **Scale repos wisely** — Right strategy, tools, and structure.
- **Rotate leaked secrets** — Remove from history, rotate, prevent repeat.

AZ-400

DevOps Engineer — Visual Notes

Part 4: Build & Release Pipelines — Foundations

vighnesh-notes
vighneshnaik.com
linkedin.com/in/vighneshnaik6

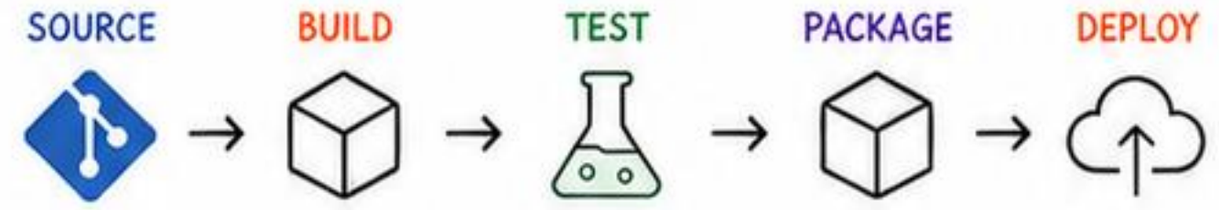
1 WHY PIPELINES MATTER

- Automate delivery end-to-end from code to value.
- Increase quality, speed, and repeatability.
- Enable collaboration and feedback loops.
- Key to DevOps culture and continuous improvement.

WEIGHTING REMINDER

50–55%
Build & Release Pipelines
★ HIGHEST WEIGHT

2 CI/CD FLOW OVERVIEW



Automate everything you can • Secure by design • Measure what matters
Make changes small & reversible • Collaborate & communicate continuously

SAMPLE PIPELINE (CONCEPTUAL)

- trigger: push
- stages:
  - build ✓
  - test ✓
  - package ✓
  - deploy-dev ✓
  - deploy-prod ✓

Tip: Keep pipelines fast, reliable, and easy to understand!

3 YAML FOUNDATIONS

```yaml
trigger:
  branches:
    include:
      - main
stages:
- stage: Build
  jobs:
  - job: BuildJob
    steps:
    - task: UseDotNet@2
      inputs:
        packageType: sdk
        version: '8.x'
    - script: dotnet build
- stage: Test
  dependsOn: Build
  jobs:
  - job: TestJob
    steps:
    - script: dotnet test
      displayName: Run tests
```

- **Triggers** — When the pipeline runs (push, PR, schedule, etc.)
- **Stages** — Group of jobs with a goal
- **Jobs** — Run on an agent (parallel or sequential)
- **Steps** — Ordered set of actions
- **Tasks** — Prebuilt or custom operations
- **Templates** — Reusable YAML snippets
- **Parameters** — Input values at queue time
- **Variables** — Values used in the pipeline
- **Variable Groups** — Group & share variables securely
- **Conditions** — Run steps/jobs when expressions are true
- **Dependencies** — Control order & access outputs

4 GITHUB ACTIONS vs AZURE PIPELINES

WHAT THEY ARE

GitHub Actions — CI/CD service deeply integrated with GitHub.

Azure Pipelines — Microsoft service for end-to-end CI/CD across any platform.

COMMON SIMILARITIES

- YAML-driven pipelines
- Support for builds, tests, deploys
- Artifacts & packages
- Secrets management
- Environments & approvals
- Extensible with tasks/actions
- Concurrency & caching capabilities

WHEN EACH IS USED

GitHub Actions
- Code hosted on GitHub
- Simple to medium pipelines
- Strong ecosystem of actions
- GitHub-first workflows

Azure Pipelines
- Code in Azure Repos, GitHub, or other Git
- Complex/enterprise scenarios
- Deep Azure integrations
- Advanced approvals & compliance needs

5 RUNNERS AND AGENTS

MICROSOFT-HOSTED
- Managed by Microsoft
- Always up to date
- Easy & fast to use
- Public network
- Good for most scenarios

SELF-HOSTED
- You manage & maintain
- Custom tools & configs
- Private network access
- Scale as you need
- More responsibility

KEY CONSIDERATIONS

Networking (public vs private) | Security & secrets | Maintenance & patching | Cost & scale | Availability & reliability

6 PACKAGES AND ARTIFACTS

WHERE TO STORE

GitHub Packages — Registry for containers, NuGet, npm, Maven, Helm, and more.

Azure Artifacts — Feeds for NuGet, Maven, Python, npm, Universal, and more.

FEEDS & VIEWS

- Private & public feeds
- Upstream sources (proxy NuGet.org, PyPI.org, etc.)
- Views for curated subsets (Release, Preview, Internal)

ARTIFACT VERSIONING

Immutability + clear naming = traceability & rollbacks

VERSIONING BEST PRACTICES

SemVer (1.2.3) | CalVer (2025.05.12)

7 TESTING INSIDE PIPELINES

- **Unit Tests** — Fast feedback on code (c#, xUnit, NUnit, pytest)
- **Integration Tests** — Validate components working together
- **UI / End-to-End Tests** — Validate user journeys (e.g., Playwright, Cypress, Selenium)
- **Load Tests** — Validate performance & scale (e.g., k6, JMeter, Azure Load Testing)
- **Code Coverage** — Measure test coverage (e.g., Coverlet, JaCoCo)
- **Quality Gates** — Fail the pipeline if thresholds are not met

QUALITY GATE EXAMPLE

- Min coverage ≥ 80%
- 0 critical vulnerabilities
- All tests passed
- No high severity issues

Fail early. Fix fast. Ship with confidence.

8 PIPELINE HEALTH

- **Flaky Tests** — Stabilize tests, quarantine, and track failures.
- **Caching** — Cache dependencies & tools to speed up builds.
- **Performance** — Profile & optimize slow steps, parallelize where possible
- **Reliability** — Tests transient failures, use idempotent steps
- **Retention** — Keep logs, artifacts, and releases for the right time
- **Optimization** — Limit parallel runs, cancel superseded runs
- **Optimization** — Right-size agents, reuse work, avoid redoing work.

QUICK WINS

- Cache restore early
- Run tests in parallel
- Use shallow clones
- Publish only what's needed
- Clean up artifacts

9 EXAM CLUES & COMMON SCENARIOS

- **SPEED UP Pipeline** — Use caching, parallel jobs, right agent, avoid unnecessary steps.
- **SECURE Pipeline** — Use secret/variable groups, limit privilege, secure tasks,
- **RELIABLE Pipeline** — Handle flaky tests, retries, timeouts, idempotent tasks.
- **REDUCE COSTS** — Use self-hosted runners, right agent size, retention policies.
- **TRACEABILITY** — Use versioned artifacts, releases, metadata, logging.

SCENARIO EXAMPLES

- ☑ Design a pipeline for multi-stage deployment with approvals.
- ☑ Choose between GitHub Actions and Azure Pipelines.
- ☑ Configure variable groups and secrets securely.
- ☑ Improve pipeline performance and reliability.
- ☑ Implement testing and quality gates.

Read the scenario → Identify goals & constraints
Apply best practices → Choose the best option

Stay consistent, practice hands-on, and think like a DevOps engineer.
Build. Test. Automate. Repeat. You've got this!

Clearly smart. Practice hands-on.
You've got the skills — go own it!

You've got this!

AZ-400 DevOps Engineer — Visual Notes

Part 5: Pipelines Delivery, Deployment & IaC

5/7

vighnesh-notes
vighneshnaik.com
linkedin.com/in/vighneshnaik6

DOMAIN 3 (Part 2): Build and Release Pipelines

1 ENVIRONMENTS, APPROVALS, CHECKS, GATES & DEPLOYMENT HISTORY

Environment Flow

Dev → Test → Staging → Prod

Approvals

- Manual approvals
- Required reviewers
- Email & Teams notifications

Checks

- Pre-deployment checks
- Query & REST checks
- Business hours checks
- Invoke Azure Function

Gates

- Quality gates
- Azure Monitor alerts
- Work item state
- Test results

Deployment History

- View audit trail, logs, artifacts, approvals & changes
- Roll back to a previous successful release

2 DEPLOYMENT STRATEGIES & TECHNIQUES

Blue-Green

Swap traffic between prod (blue) & green environment.

Rolling Updates

Update instances in batches to keep service available.

Canary

Release to a small subset of users first.

Deployment Slots / Swaps (App Service)

SLOT → PROD

Deploy to slot, test, then swap with production.

Ring / Progressive Exposure

Expand release across rings (internal → external).

Feature Flags

Toggle features on/off without new deployment.

A/B Testing

A B

Route traffic to A or B versions and compare.

Hotfix & Rollback

Quick fix via hotfix branch or roll back to last good release.

3 DEPENDENCIES & RELEASE ORCHESTRATION

APP → DB

App + Database Coordination

Deploy app and DB changes together safely.

Dependency Order

Deploy in correct sequence (DB → API → Web / Clients).

Zero-Downtime Thinking

Design releases to avoid downtime & data loss.

Orchestration Patterns

- Single pipeline with stages
- Multi-pipeline with resource dependencies
- Release gates to control flow
- Artifacts, variables & output variables for coordination
- Use Environments & approvals for critical dependencies

4 INFRASTRUCTURE AS CODE (IaC)

ARM Templates

Native Azure templates (JSON) for resources.

Bicep (Preferred)

Declarative language for simpler, reusable Azure IaC.

Terraform (Open Source)

Multi-cloud IaC tool. Works with Azure.

Azure Automation State Configuration

Automate OS configuration (DSC) for consistency.

Azure Machine Configuration

Manage VMs/VMSS images and OS settings at scale.

Azure Deployment Environments

Model and manage target environments as code.

Treat infrastructure like code: version, review, test & deploy automatically.

5 CONFIGURATION & SECRET HANDLING IN DELIVERY

Variable Groups

Centralize variables & secrets for reuse across pipelines.

Key Vault Integration

Store secrets in Azure Key Vault and reference in pipelines.

Library & Template Variables

Share configuration values across pipelines.

Parameterization

Use parameters in templates for environment-specific values.

6 SAFE RELEASE PRACTICES

Small Batches

Release gradually to reduce risk.

Automated Validation

Run automated tests, scans, linting, and smoke tests.

Post-Deployment Verification

Health checks, synthetic tests, monitoring alerts.

Observability Hooks

Enable logs, metrics, traces from day one.

7 MAINTENANCE & CONTINUOUS IMPROVEMENT

Classic to YAML Migration

Move from Classic pipelines to YAML for flexibility and version control.

Template Reuse

Use templates & component templates to avoid copy-paste & ensure consistency.

Standardization

Standardize stages, tasks, naming, variables and environments.

Cost & Time Optimization

Use parallel jobs, caching, self-hosted agents, and right-size resources.

8 EXAM TIPS / BEST-ANSWER PATTERNS

Always Read the Scenario Carefully

Identify the goal (speed, safety, cost, compliance).

Think Azure-Native First

Prefer Azure DevOps, Azure Services & managed options.

Security First

Use least privilege, approvals, checks, and secret management.

Choose the Best Fit

Consider trade-offs: speed vs risk, cost vs scale.

YAML > Classic

Remove incorrect or not-ideal options first.

YAML > Classic

Choose YAML pipelines for flexibility, versioning & CI/CD best practices.

QUICK RECAP – DOMAIN 3 (PART 2)

- Control releases with approvals, checks & gates
- Choose the right deployment strategy
- Orchestrate dependencies safely
- Use IaC for consistent & repeatable infra
- Manage config & secrets securely
- Validate, monitor & learn continuously
- Optimize & standardize pipelines

Stay consistent, automate everything, and think like a DevOps engineer.
Build. Test. Learn. Repeat.

You've got this!

AZ-400 DevOps Engineer — Visual Notes

Part 6: Security & Compliance

DOMAIN 4: SECURITY AND COMPLIANCE (10–15%)

1 SECURITY MINDSET

- **Least privilege** — Grant only the access that's needed.
- **Secure by design** — Build security in from the start.
- **Defense in depth** — Use multiple layers of protection.
- **Auditability** — Ensure actions are traceable and reviewable.

2 IDENTITY CHOICES

- **Azure AD / Microsoft Entra** — Central identity & access management.
- **Managed Identities** — Apps authenticate to Azure without storing credentials.
  - **System-assigned** — Tied to one resource's lifecycle.
  - **User-assigned** — Reusable across resources.
- **Service Principals** — App identities for automation.
- **Service Connections** — Secure links to external systems.

3 GITHUB AND AZURE DEVOPS AUTHENTICATION

- **GitHub Apps** — Recommended for first-party integrations.
- **GITHUB_TOKEN** — Short-lived token for workflow jobs; scoped to permissions.
- **Personal Access Tokens (PATs)** — Use sparingly; store securely.
- **Access Levels** — Repo, Org, Enterprise.
- **Teams & Groups** — Manage access at scale; use least privilege.

4 SECRETS MANAGEMENT

- **Azure Key Vault** — Store secrets, keys, and certificates securely.
- **Secure Files** — Upload and use encrypted files in pipelines.
- **Variable Groups** — Centralize variables; link to Key Vault for secrets.
- **Secret Masking** — Prevent secrets from appearing in logs.
- **Avoid hard-coded secrets** — Never commit secrets to code.

5 SECRETLESS CLOUD ACCESS

- **Workload Identity Federation (OIDC)** — Eliminate long-lived secrets for cloud access.

GitHub / Azure DevOps → OIDC Token (Short-lived) → Azure / Cloud Resource

- ✓ Trust relationship via OIDC
- ✓ Short-lived tokens
- ✓ Scoped to specific resources
- ✓ Revocable and auditable

6 SECURE PIPELINES

- **Permissions** — Use least privilege for service connections, agents, and tasks.
- **Log Hygiene** — Mask secrets and avoid echoing sensitive values.
- **Prevent leakage** — Secure scripts, artifacts, and environment variables.
- **Approvals & gates** — Manual approvals for risky changes.
- **Segregation of duties** — Separate build, release, and approval roles.

7 SECURITY SCANNING

- **Dependency scanning** — Find vulnerable open-source dependencies.
- **Secret scanning** — Detect leaked secrets and keys.
- **SAST** — Static analysis of source code.
- **Container scanning** — Scan container images for vulnerabilities.
- **License scanning** — Detect license risks and compliance issues.
- **CodeQL** — Semantic code analysis.
- **Dependabot** — Automated dependency updates & alerts.
- **Defender for Cloud DevOps Security** — End-to-end DevSecOps posture.
- **GitHub Advanced Security** — SAST, secret scanning, CodeQL, and more.

8 COMPLIANCE & GOVERNANCE REMINDERS

- **Know compliance requirements** (e.g., GDPR, HIPAA, SOC 2, ISO).
- **Apply policies as code** — Use Azure Policy, GitHub rulesets, and branch protection.
- **Data protection** — Encrypt in transit and at rest.
- **Retention & auditing** — Keep logs, artifacts, and audit trails.
- **Access reviews** — Periodically review and revoke unneeded access.
- **Document & report** — Maintain evidence and compliance documentation.

9 EXAM TIPS

- ★ Know identity options and when to use each.
- ★ Understand secretless access with OIDC.
- ★ Choose the right authentication method (GitHub Apps > PATs).
- ★ Secure your pipelines: permissions, approvals, masking.
- ★ Know security scanning tools and their purposes.
- ★ Auditability and governance are key themes.

Stay consistent, practice hands-on, and think like a DevOps engineer.

Build. Test. Learn. Repeat.

✓ You've got this! 😀

AZ-400 DevOps Engineer – Visual Notes

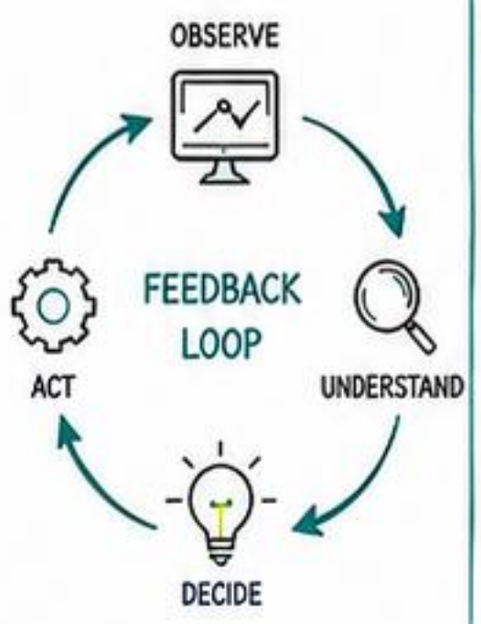
Part 7: Instrumentation & Observability

7/7


vighnesh-notes
vighneshnaik.com
linkedin.com/in/vighneshnaik6

1 WHY OBSERVABILITY MATTERS / FEEDBACK LOOP

- See what's happening across your systems
- Detect issues early
- Understand impact and root cause
- Improve performance and reliability
- Drive continuous improvement



2 CORE TELEMETRY TYPES

METRICS
- Numeric time-series
- Aggregated over time
- Counters, gauges, rates

LOGS
- Discrete events
- Rich, flexible data
- Great for troubleshooting and audit

DISTRIBUTED TRACES
- Follow requests across services
- Spans, latency, dependencies
- End-to-end visibility

3 AZURE OBSERVABILITY TOOLS

Azure Monitor
Central platform for metrics, logs, alerts, dashboards, and workbooks

Log Analytics
Collect, store, query logs with KQL
Powerful analytics at scale

Application Insights
Application performance monitoring
Traces, requests, failures, users

VM Insights
Health and performance of VMs
Process, dependencies, updates

Container Insights
Kubernetes and container monitoring
Performance, logs, health

Storage & Network Insights
Storage metrics and capacity
Network performance, flow, latency

4 GITHUB INSIGHTS & PIPELINE MONITORING

GitHub Insights
- DORA metrics (Deployment freq, Lead time, MTTR, Change fail rate)
- Code health, PR insights
- Team performance

Pipeline Monitoring
- Track runs, stages, tasks
- Duration, success rate, failures
- Timeline, logs, artifacts
- Alerts on failures or delays

- Email, Teams, Webhooks
- Thresholds, conditions
Escalation paths, on-call

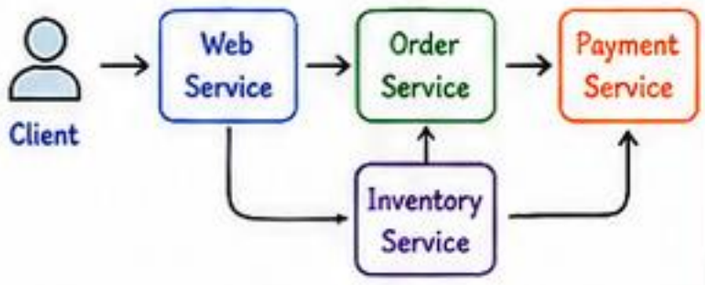
5 INFRASTRUCTURE SIGNALS

CPU	• Utilization % • Ready time
MEMORY	• Utilization % • Available memory
DISK	• IOPS / Throughput • Latency / Queue length • Free space
NETWORK	• In/Out throughput • Latency / Packet loss
SATURATION	• resource pressure • Throttling, limits • Queues & backlogs

6 APPLICATION SIGNALS

REQUESTS	• Request rate • Throughput
LATENCY	• Response time • Percentiles (P50/P95/P99)
FAILURES	• Error rate • Exception count
DEPENDENCIES	• Dependency health • External service calls
TRAFFIC	• Users / Sessions • Geographic / Devices
ERRORS	• Exceptions, stack traces • Error trends
SATURATION	• Concurrency, thread pool • Connection pool, queues
USER BEHAVIOR	• Page views, clicks, funnels • User flows, engagement

7 DISTRIBUTED TRACING & CORRELATION



- Each request has a trace ID
- Spans show timing and dependencies
Correlate logs, metrics, and traces
- Identify bottlenecks and failures

Result: Faster root cause, better reliability

8 KQL BASICS (LOG ANALYTICS)

```
// Filter
 requests
 | where success == false
 | where timestamp > ago(1h)

// Summarize
 | summarize count() by resultCode

// Aggregate over time
 | summarize avg(duration) by bin(timestamp, 5m)
```

FILTER	SUMMARIZE	AGGREGATE	TIME ANALYSIS
where, has, stratswith	count, sum, avg, min, max	by, bin(), distinct count	ago(), bin(), make-series

Tip: Start with filter → summarize → visualize

9 ALERTS, DASHBOARDS, WORKBOOKS & ACTIONABLE FEEDBACK LOOPS

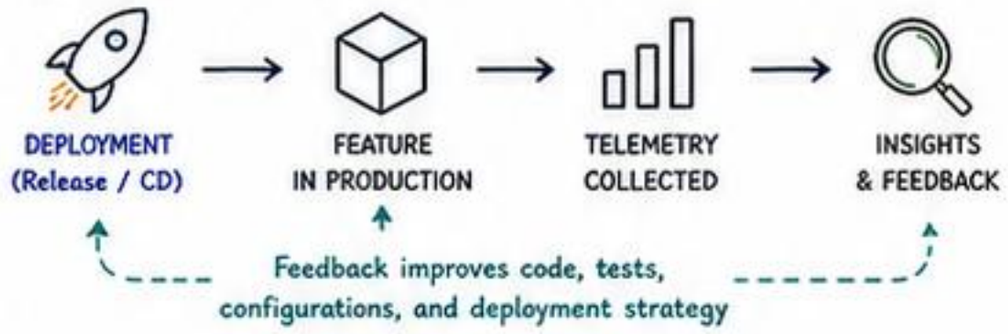
Alerts
- Metric or log alert rules
- Thresholds, dynamic baselines
- Action groups, notifications

Dashboards
- Real-time overview of key signals
- Share & pin what matters

Workbooks
- Interactive visual reports
- Drill down with KQL
- Parameterized & shareable

Act → Investigate → Resolve → Learn → Improve
Capture learnings and update dashboards, alerts, runbooks, and tests.

10 HOW DEPLOYMENTS CONNECT TO TELEMETRY



Close the loop for continuous improvement

11 EXAM TIPS (DOMAIN 5: INSTRUMENTATION STRATEGY)

- Know core telemetry types and what they're best for
- Understand Azure Monitor and related insight tools
- Recognize key infrastructure and application signals
- Know how tracing and correlation speed up troubleshooting
- Write simple KQL: filter, summarize, aggregate, time bins
- Know how alerts, dashboards, and workbooks are used
- Understand feedback loops and continuous improvement

REMEMBER:
Good observability turns data into insight, insight into action, and action into reliability.

Stay consistent, practice hands-on, and think like a DevOps engineer.

You've got this!