

AZ-400T00: Designing and Implementing Microsoft DevOps solutions

Duration : 32 Hours

Audience profile

As a DevOps engineer, you're a developer or infrastructure administrator who also has subject matter expertise in working with people, processes, and products to enable continuous delivery of value in organizations.

Your responsibilities for this role include delivering Microsoft DevOps solutions that provide continuous security, integration, testing, delivery, deployment, monitoring, and feedback. You design and implement flow of work, collaboration, communication, source control, and automation.

As a DevOps engineer, you work on cross-functional teams that include:

- Developers
- Site reliability engineers
- Azure administrators
- Security engineers

You must have experience both administering and developing in Azure, with strong skills in at least one of these areas. You should also have experience implementing both GitHub and Azure DevOps solutions.

Skills at a glance

- Design and implement processes and communications (10–15%)
- Design and implement a source control strategy (10–15%)
- Design and implement build and release pipelines (50–55%)
- Develop a security and compliance plan (10–15%)
- Implement an instrumentation strategy (5–10%)

Design and implement processes and communications (10–15%)

Design and implement traceability and flow of work

- Design and implement a structure for the flow of work, including GitHub Flow
- Design and implement a strategy for feedback cycles, including notifications and GitHub issues

- Design and implement integration for tracking work, including GitHub projects, Azure Boards, and repositories
- Design and implement source, bug, and quality traceability

Design and implement appropriate metrics and queries for DevOps

- Design and implement a dashboard, including flow of work, such as cycle times, time to recovery, and lead time
- Design and implement appropriate metrics and queries for project planning
- Design and implement appropriate metrics and queries for development
- Design and implement appropriate metrics and queries for testing
- Design and implement appropriate metrics and queries for security
- Design and implement appropriate metrics and queries for delivery
- Design and implement appropriate metrics and queries for operations

Configure collaboration and communication

- Document a project by configuring wikis and process diagrams, including Markdown and Mermaid syntax
- Configure release documentation, including release notes and API documentation
- Automate creation of documentation from Git history
- Configure integration by using webhooks
- Configure integration between Azure Boards and GitHub repositories
- Configure integration between GitHub or Azure DevOps and Microsoft Teams

Design and implement a source control strategy (10–15%)

Design and implement branching strategies for the source code

- Design a branch strategy, including trunk-based, feature branch, and release branch
- Design and implement a pull request workflow by using branch policies and branch protections
- Implement branch merging restrictions by using branch policies and branch protections

Configure and manage repositories

- Design and implement a strategy for managing large files, including Git Large File Storage (LFS) and git-fat
- Design a strategy for scaling and optimizing a Git repository, including Scalar and cross-repository sharing
- Configure permissions in the source control repository
- Configure tags to organize the source control repository
- Recover specific data by using Git commands
- Remove specific data from source control

Design and implement build and release pipelines (50–55%)

Design and implement a package management strategy

- Recommend package management tools including GitHub Packages registry and Azure Artifacts
- Design and implement package feeds and views for local and upstream packages
- Design and implement a dependency versioning strategy for code assets and packages, including semantic versioning (SemVer) and date-based (CalVer)
- Design and implement a versioning strategy for pipeline artifacts

Design and implement a testing strategy for pipelines

- Design and implement quality and release gates, including security and governance
- Design a comprehensive testing strategy, including local tests, unit tests, integration tests, and load tests
- Implement tests in a pipeline, including configuring test tasks, configuring test agents, and integration of test results
- Implement code coverage analysis

Design and implement pipelines

- Select a deployment automation solution, including GitHub Actions and Azure Pipelines
- Design and implement a GitHub runner or Azure DevOps agent infrastructure, including cost, tool selection, licenses, connectivity, and maintainability
- Design and implement integration between GitHub repositories and Azure Pipelines

- Develop and implement pipeline trigger rules
- Develop pipelines by using YAML
- Design and implement a strategy for job execution order, including parallelism and multi-stage pipelines
- Develop and implement complex pipeline scenarios, such as hybrid pipelines, VM templates, and self-hosted runners or agents
- Create reusable pipeline elements, including YAML templates, task groups, variables, and variable groups
- Design and implement checks and approvals by using YAML-based environments

Design and implement deployments

- Design a deployment strategy, including blue-green, canary, ring, progressive exposure, feature flags, and A/B testing
- Design a pipeline to ensure that dependency deployments are reliably ordered
- Plan for minimizing downtime during deployments by using virtual IP address (VIP) swap, load balancing, rolling deployments, and deployment slot usage and swap
- Design a hotfix path plan for responding to high-priority code fixes
- Design and implement a resiliency strategy for deployment
- Implement feature flags by using Azure App Configuration Feature Manager
- Implement application deployment by using containers, binaries, and scripts
- Implement a deployment that includes database tasks

Design and implement infrastructure as code (IaC)

- Recommend a configuration management technology for application infrastructure
- Implement a configuration management strategy for application infrastructure
- Define an IaC strategy, including source control and automation of testing and deployment
- Design and implement desired state configuration for environments, including Azure Automation State Configuration, Azure Resource Manager, Bicep, and Azure Automate Machine Configuration
- Design and implement Azure Deployment Environments for on-demand self-deployment

Maintain pipelines

- Monitor pipeline health, including failure rate, duration, and flaky tests
- Optimize a pipeline for cost, time, performance, and reliability
- Optimize pipeline concurrency for performance and cost
- Design and implement a retention strategy for pipeline artifacts and dependencies
- Migrate a pipeline from classic to YAML in Azure Pipelines

Develop a security and compliance plan (10–15%)

Design and implement authentication and authorization methods

- Choose between Service Principals and Managed Identity (including system-assigned and user-assigned)
- Implement and manage GitHub authentication, including GitHub Apps, GITHUB_TOKEN, and personal access tokens
- Implement and manage Azure DevOps service connections and personal access tokens
- Design and implement permissions and roles in GitHub
- Design and implement permissions and security groups in Azure DevOps
- Recommend appropriate access levels, including stakeholder access in Azure DevOps and outside collaborator access in GitHub
- Configure projects and teams in Azure DevOps

Design and implement a strategy for managing sensitive information in automation

- Implement and manage secrets, keys, and certificates by using Azure Key Vault
- Implement and manage secrets in GitHub Actions and Azure Pipelines
- Design and implement a strategy for managing sensitive files during deployment, including Azure Pipelines secure files
- Design pipelines to prevent leakage of sensitive information

Automate security and compliance scanning

- Design a strategy for security and compliance scanning, including dependency, code, secret, and licensing scanning
- Configure Microsoft Defender for Cloud DevOps Security
- Configure GitHub Advanced Security for both GitHub and Azure DevOps

- Integrate GitHub Advanced Security with Microsoft Defender for Cloud
- Automate container scanning, including scanning container images and configuring an action to run CodeQL analysis in a container
- Automate analysis of licensing, vulnerabilities, and versioning of open-source components by using Dependabot alerts

Implement an instrumentation strategy (5–10%)

Configure monitoring for a DevOps environment

- Configure Azure Monitor and Log Analytics to integrate with DevOps tools
- Configure collection of telemetry by using Application Insights, VM Insights, Container Insights, Storage Insights, and Network Insights
- Configure monitoring in GitHub, including enabling insights and creating and configuring charts
- Configure alerts for events in GitHub Actions and Azure Pipelines

Analyze metrics from instrumentation

- Inspect infrastructure performance indicators, including CPU, memory, disk, and network
- Analyze metrics by using collected telemetry, including usage and application performance
- Inspect distributed tracing by using Application Insights
- Interrogate logs using basic Kusto Query Language (KQL) queries