AZ-400
Microsoft Azure DevOps Solutions

Exam number: AS-400

Exam title: Microsoft Azure DevOps Solutions

Language(s) this exam will be available in: English

Audience (IT professionals, Developers, Information workers, etc.): IT Professionals

Technology: Microsoft Azure

Exam provider (VUE, Certiport, or both)**:** **VUE**

Exam Design

Audience Profile

Candidates for this exam are DevOps professionals who combine people, process, and technologies to continuously deliver valuable products and services that meet end user needs and business objectives. DevOps professionals streamline delivery by optimizing practices, improving communications and collaboration, and creating automation. They design and implement strategies for application code and infrastructure that allow for continuous integration, continuous testing, continuous delivery, and continuous monitoring and feedback.

Candidates must be proficient with Agile practices. They must be familiar with both Azure administration and Azure development and experts in at least one of these areas. Azure DevOps professionals must be able to design and implement DevOps practices for version control, compliance, infrastructure as code, configuration management, build, release, and testing by using Azure technologies.

Candidates for this exam must be proficient with Microsoft Azure DevOps technologies and have basic knowledge of common third-party DevOps tools used on Azure (e.g., Jenkins, Terraform, Chef, Ansible, etc.).

Skills Measured

Note: This document shows tracked changes that are effective as of March 15, 2019.

Design a DevOps Strategy (20-25%)

Recommend a migration and consolidation strategy for DevOps tools

- Analyze existing artifact (e.g., deployment packages, NuGet) and container repositories
- Analyze existing test management tools
- Analyze existing work management tools
- Recommend migration and integration strategies for artifact repositories, source control, test management, and work management

Design and implement an Agile work management approach

- Identify and recommend project metrics, KPIs, and DevOps measurements (e.g., cycle time, lead time, WIP limit)
- Implement tools and processes to support Agile work management
- Mentor team members on Agile techniques and practices
- Recommend an organization structure that supports scaling Agile practices
- Recommend in-team and cross-team collaboration mechanisms

Design a quality strategy

- Analyze existing quality environment
- Identify and recommend quality metrics
- Recommend a strategy for feature flag lifecycle
- Recommend a strategy for measuring and managing technical debt
- Recommend changes to team structure to optimize quality
- Recommend performance testing strategy

Design a secure development process

- Inspect and validate code base for compliance
- Inspect and validate infrastructure for compliance
- Recommend a secure development strategy
- Recommend tools and practices to integrate code security validation (e.g., static code analysis)
- Recommend tools and practices to integrate infrastructure security validation

Design a tool integration strategy

- Design a license management strategy (e.g., VSTS users, concurrent pipelines, test environments, open source software licensing, third-party DevOps tools and services, package management licensing)
- Design a strategy for end-to-end traceability from work items to working software
- Design a strategy for integrating monitoring and feedback to development teams
- Design an authentication and access strategy
- Design a strategy for integrating on-premises and cloud resources

Implement DevOps Development Processes (20-25%)

Design a version control strategy

- Recommend branching models
- Recommend version control systems
- Recommend code flow strategy

Implement and integrate source control

- Integrate external source control
- Integrate source control into third-party continuous integration and continuous deployment (CI/CD) systems

Implement and manage build infrastructure

- Implement private and hosted agents
- Integrate third party build systems
- Recommend strategy for concurrent pipelines
- Manage Azure pipeline configuration (e.g., agent queues, service endpoints, pools, webhooks)

Implement code flow

- Implement pull request strategies
- Implement branch and fork strategies
- Configure branch policies

Implement a mobile DevOps strategy

- Manage mobile target device sets and distribution groups
- Manage target UI test device sets
- Provision tester devices for deployment
- Create public and private distribution groups

Managing application configuration and secrets

- Implement a secure and compliant development process
- Implement general (non-secret) configuration data
- Manage secrets, tokens, and certificates
- Implement applications configurations (e.g., Web App, Azure Kubernetes Service, containers)
- Implement secrets management (e.g., Web App, Azure Kubernetes Service, containers, Azure Key Vault)
- Implement tools for managing security and compliance in the pipeline

Implement Continuous Integration (10-15%)

Manage code quality and security policies

- Monitor code quality
- Configure build to report on code coverage
- Manage automated test quality
- Manage test suites and categories
- Monitor quality of tests
- Integrate security analysis tools (e.g., SonarQube, WhiteSource Bolt, Open Web Application Security Project)

Implement a container build strategy

Create deployable images (e.g., Docker, Hub, Azure Container Registry)
Analyze and integrate Docker multi-stage builds

Implement a build strategy

Design build triggers, tools, integrations, and workflow
Implement a hybrid build process
Implement multi-agent builds
Recommend build tools and configuration (e.g. Azure Pipelines, Jenkins)
Set up an automated build workflow

Implement Continuous Delivery (10-15%)

Design a release strategy

Recommend release tools
Identify and recommend release approvals and gates
Recommend strategy for measuring quality of release and release process
Recommend strategy for release notes and documentation
Select appropriate deployment pattern

Set up a release management workflow

Automate inspection of health signals for release approvals by using release gates
Configure automated integration and functional test execution
Create a release pipeline (e.g., Azure Kubernetes Service, Service Fabric, WebApp)
Create multi-phase release pipelines
Integrate secrets with release pipeline
Provision and configure environments
Manage and modularize tasks and templates (e.g., task and variable groups)

Implement an appropriate deployment pattern

Implement blue-green deployments
Implement canary deployments
Implement progressive exposure deployments
Scale a release pipeline to deploy to multiple endpoints (e.g., deployment groups, Azure Kubernetes Service, Service Fabric)

Implement Dependency Management (5-10%)

Design a dependency management strategy

Recommend artifact management tools and practices (Azure Artifacts, npm, maven, Nuget)
Abstract common packages to enable sharing and reuse
Inspect codebase to identify code dependencies that can be converted to packages
Identify and recommend standardized package types and versions across the solution
Refactor existing build pipelines to implement version strategy that publishes packages

Manage security and compliance

Inspect open source software packages for security and license compliance to align with corporate standards (e.g., GPLv3)
Configure build pipeline to access package security and license rating (e.g., Black Duck, White Source)

Configure secure access to package feeds

Implement Application Infrastructure (15-20%)

Design an infrastructure and configuration management strategy

Analyze existing and future hosting infrastructure
Analyze existing Infrastructure as Code (IaC) technologies
Design a strategy for managing technical debt on templates
Design a strategy for using transient infrastructure for parts of a delivery lifecycle
Design a strategy to mitigate infrastructure state drift

Implement Infrastructure as Code (IaC)

Create nested resource templates
Manage secrets in resource templates
Provision Azure resources
Recommend an Infrastructure as Code (IaC) strategy
Recommend appropriate technologies for configuration management (ARM Templates, Terraform, Chef, Puppet, Ansible)

Manage Azure Kubernetes Service infrastructure

Provision Azure Kubernetes Service (e.g., using ARM templates, CLI)
Create deployment file for publishing to Azure Kubernetes Service (e.g., kubectl, Helm)
Develop a scaling plan

Implement infrastructure compliance and security

Implement compliance and security scanning
Prevent drift by using configuration management tools
Automate configuration management by using PowerShell Desired State Configuration (DSC)
Automate configuration management by using a VM Agent with custom script extensions
Set up an automated pipeline to inspect security and compliance

Implement Continuous Feedback (10-15%)

Recommend and design system feedback mechanisms

Design practices to measure end-user satisfaction (e.g., Send a Smile, app analytics)
Design processes to capture and analyze user feedback from external sources (e.g., Twitter, Reddit, Help Desk)
Design routing for client application crash report data (e.g., HockeyApp)
Recommend monitoring tools and technologies
Recommend system and feature usage tracking tools

Implement process for routing system feedback to development teams

Configure crash report integration for client applications
Develop monitoring and status dashboards
Implement routing for client application crash report data (e.g., HockeyApp)
Implement tools to track system usage, feature usage, and flow
Integrate and configure ticketing systems with development team's work management system (e.g., IT Service Management connector, ServiceNow Cloud Management, App Insights work items)

Optimize feedback mechanisms

- Analyze alerts to establish a baseline
- Analyze telemetry to establish a baseline
- Perform live site reviews and capture feedback for system outages
- Perform ongoing tuning to reduce meaningless or non-actionable alerts