TechyEdz Solutions

Training | Consulting | Developement | Outsourcing

Azure DevOps (AZ-400)

9032803832

9032803832

contact@techyedz.com

www.techyedz.com

Azure - Designing and Implementing Microsoft DevOps Solutions (AZ-400)

Course Overview:

This course measures your ability to accomplish the following technical tasks: develop an instrumentation strategy; develop a Site Reliability Engineering (SRE) strategy; develop a security and compliance plan; manage source control; facilitate communication and collaboration; define and implement continuous integration; and define and implement a continuous delivery and release management strategy.

Course Outline:

Develop an Instrumentation Strategy

1. Design and implement logging

- assess and Configure a log framework
- design a log aggregation and storage strategy (e.g. Azure storage)
- design a log aggregation using Azure Monitor
- manage access control to logs (workspace-centric/resource-centric)
- integrate crash analytics (App Center Crashes, Crashlytics)

2. Design and implement telemetry

- design and implement distributed tracing
- inspect application performance indicators
- inspect infrastructure performance indicators
- define and measure key metrics (CPU, memory, disk, network)
- implement alerts on key metrics (email, SMS, webhooks, Teams/Slack)
- integrate user analytics (e.g. Application Insights funnels, Visual Studio App Center, TestFlight, Google Analytics)

3. Integrate logging and monitoring solutions

- configure and integrate container monitoring (Azure Monitor, Prometheus, etc.)
- configure and integrate with monitoring tools (Azure Monitor Application Insights, Dynatrace, New Relic, Naggios, Zabbix)
- create feedback loop from platform monitoring tools (e.g. Azure Diagnostics VM extensions, Azure Platform Logs, Event Grid)

- manage Access control to the monitoring platform

Develop a Site Reliability Engineering (SRE) strategy

1. Develop an actionable alerting strategy

- identify and recommend metrics on which to base alerts
- implement alerts using appropriate metrics
- implement alerts based on appropriate log messages
- implement alerts based on application health checks
- analyze combinations of metrics
- develop communication mechanism to notify users of degraded systems
- implement alerts for self-healing activities (e.g. scaling, failovers)

2. Design a failure prediction strategy

- analyze behavior of system with regards to load and failure conditions
- calculate when a system will fail under various conditions
- measure baseline metrics for system
- recommend the appropriate tools for a failure prediction strategy

3. Design and implement a health check

- analyze system dependencies to determine which dependency should be included in health check
- calculate healthy response timeouts based on SLO for the service
- design approach for partial health situations
- integrate health check with compute environment
- implement different types of health checks (liveness, startup, shutdown)

Develop a security and compliance plan

1. Design an authentication and authorization strategy

- design an access solution (Azure AD Privileged Identity Management (PIM), Azure AD Conditional Access, MFA)
- organize the team using Azure AD groups
- implement Service Principals and Managed Identity
- configure service connections

2. Design a sensitive information management strategy

- evaluate and configure vault solution (Azure Key Vault, Hashicorp Vault)
- generate security certificates
- design a secrets storage and retrieval strategy
- formulate a plan for deploying secret files as part of a release

3. Develop security and compliance

- automate dependencies scanning for security (container scanning, OWASP)
- automate dependencies scanning for compliance (licenses: MIT, GPL)
- assess and report risks
- design a source code compliance solution (e.g. GitHub security, pipeline-based scans, Git hooks, SonarQube)

4. Design governance enforcement mechanisms

- implement Azure policies to enforce organizational requirements
- implement container scanning (e.g. static scanning, malware, crypto mining)
- design and implement Azure Container Registry Tasks (eg. Azure Policy)
- design break-the-glass strategy for responding to security incidents

Manage source control

1. Develop a modern source control strategy

- integrate/migrate disparate source control systems (e.g. GitHub, Azure Repos)
- design authentication strategies
- design approach for managing large binary files (e.g. Git LFS)
- design approach for cross repository sharing (e.g. Git sub-modules, packages)
- implement workflow hooks

2. Plan and implement branching strategies for the source code

- define Pull Requests (PR) guidelines to enforce work item correlation
- implement branch merging restrictions (e.g. branch policies, branch protections, manual, etc.)
- define branch strategy (e.g. trunk based, feature branch, release branch, GitHub flow)
- design and implement a PR workflow (code reviews, approvals)
- enforce static code analysis for code-quality consistency on PR

3. Configure repositories

- configure permissions in the source control repository
- organize the repository with git-tags
- plan for handling oversized repositories
- plan for content recovery in all repository states
- purge data from source control

4. Integrate source control with tools

- integrate GitHub with DevOps pipelines
- integrate GitHub with identity management solutions (Azure AD)
- design for GitOps
- design for ChatOps
- integrate source control artifacts for human consumption (e.g. Git changelog)

Facilitate communication and collaboration

1. Communicate deployment and release information with business stakeholders

- create dashboards combining boards, pipelines (custom dashboards on Azure DevOps)
- design a cost management communication strategy
- integrate release pipeline with work item tracking (e.g. AZ DevOps, Jira)
- integrate GitHub as repository with Azure Boards
- communicate user analytics

2. Generate DevOps process documentation

- design onboarding process for new employees
- assess and document external dependencies (e.g. integrations, packages)
- assess and document artifacts (version, release notes)

3. Automate communication with team members

- integrate monitoring tools with communication platforms (e.g. Teams, Slack, dashboards)
- notify stakeholders about key metrics, alerts, severity using communication platforms (e.g. Email, SMS, Slack, Teams)

- integrate build and release with communication platforms (e.g. build fails, release fails)
- Define and implement continuous integration

4. Design build automation

- integrate the build pipeline with external tools (e.g., Dependency and security scanning, Code coverage)
- implement quality gates (e.g. code coverage, internationalization, peer review)
- design a testing strategy (e.g. integration, load, fuzz, API, chaos)
- integrate multiple tools (e.g. GitHub Actions, Azure Pipeline, Jenkins)

5. Design a package management strategy

- recommend package management tools (e.g. GitHub Packages, Azure Artifacts, Azure Automation Runbooks Gallery, Nuget, Jfrog, Artifactory)
- design an Azure Artifacts implementation including linked feeds
- design versioning strategy for code assets (e.g. SemVer, date based)
- plan for assessing and updating and reporting package dependencies (GitHub Automated Security Updates, NuKeeper, GreenKeeper)
- design a versioning strategy for packages (e.g. SemVer, date based)
- design a versioning strategy for deployment artifacts

6. Design an application infrastructure management strategy

- assess a configuration management mechanism for application infrastructure
- define and enforce desired state configuration for environments

7. Implement a build strategy

- design and implement build agent infrastructure (include cost, tool selection, licenses, maintainability)
- develop and implement build trigger rules
- develop build pipelines
- design build orchestration (products that are composed of multiple builds)
- integrate configuration into build process
- develop complex build scenarios (e.g. containerized agents, hybrid, GPU)

8. Maintain build strategy

- monitor pipeline health (failure rate, duration, flaky tests)
- optimize build (cost, time, performance, reliability)
- analyze CI load to determine build agent configuration and capacity
- manage pipeline health
- identify the number of agents and jobs to run in parallel
- investigate test failures

9. Design a process for standardizing builds across organization

- manage self-hosted build agents (VM templates, containerization, etc.)
- create reuseable build subsystems (YAML templates, Task Groups, Variable Groups, etc.)

Define and implement a continuous delivery and release management strategy

1. Develop deployment scripts and templates

- recommend a deployment solution (e.g. GitHub Actions, Azure Pipelines, Jenkins, CircleCI, etc.)
- design and implement Infrastructure as code (ARM, Terraform, PowerShell, CLI)
- develop application deployment process (container, binary, scripts)
- develop database deployment process (migrations, data movement, ETL)
- integrate configuration management as part of the release process
- develop complex deployments (IoT, Azure IoT Edge, mobile, App Center, DR, multi- region, CDN, sovereign cloud, Azure Stack, etc.)

2. Implement an orchestration automation solution

- combine release targets depending on release deliverable (e.g., Infrastructure, code, assets, etc.)
- design the release pipeline to ensure reliable order of dependency deployments
- organize shared release configurations and process (YAML templates, variable groups)
- design and implement release gates and approval processes

3. Plan the deployment environment strategy

- design a release strategy (blue/green, canary, ring)
- implement the release strategy (using deployment slots, load balancer configurations, Azure Traffic Manager, feature toggle, etc.)
- select the appropriate desired state solution for a deployment

environment (PowerShell DSC, Chef, Puppet, etc.)

- plan for minimizing downtime during deployments (VIP Swap, Load balancer, rolling
    - deployments, etc.)
- design a hotfix path plan for responding to high priority code fixes

Prerequisites:

- Fundamental knowledge about Azure, version control, Agile software development, and core software development principles. It would be helpful to have experience in an organization that delivers software.

- To become a Microsoft Certified: Azure DevOps Engineer Expert, you must either earn the Azure Administrator Associate or Azure Developer Associate certification.

- Great pre-requisite courses for those certifications are Microsoft Azure Administrator (AZ-104) or Developing Solutions for Microsoft Azure (AZ-204)

Who Should Attend:

- Candidates for this exam should have subject matter expertise working with people, processes, and technologies to continuously deliver business value.

- Responsibilities for this role include designing and implementing strategies for collaboration, code, infrastructure, source control, security, compliance, continuous integration, testing, delivery, monitoring, and feedback.

- A candidate for this exam must be familiar with both Azure administration and development and must be expert in at least one of these areas.

Number of Hours: 40hrs

Certification: AZ-400

Key Features:

- One to One Training
- Online Training
- Fastrack & Normal Track
- Resume Modification
- Mock Interviews
- Video Tutorials
- Materials

- Real Time Projects
- Virtual Live Experience
- Preparing for Certification