# AZ-400: DESIGNING AND IMPLEMENTING MICROSOFT DEVOPS SOLUTIONS

| | |
|---|---|
| TRAINING DATES | 18 JULY 2022 |
| TIME | TBD |
| TRAINER NAME | RAJESH KUMAR |
| NO. OF HOURS | 40 HOURS |

## AGENDA

- **Introduction to cloud computing**
- **What is AWS?**
- **What is DevOps?**
- **Transition in Software development model**
- **Waterfall -> Agile -> CI/CD -> DevOps ->**
- **What is Microsoft Azure?**
- **Microsoft Azure Services**
- **Creating a Microsoft Azure Account**
- **Azure CLI, Azure PowerShell**
- **Managing Azure Resources & Subscriptions**
- **Azure Resource Manager**
- **Microsoft Azure Architecture**

# 1 Develop an Instrumentation Strategy

## 1.1 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 and query strategy (e.g., Azure Monitor, Splunk)
- manage access control to logs (workspace-centric/resource-centric)
- integrate crash analytics (App Center Crashes, Crashlytics)

## 1.2 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)

### 1.3 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 extension, Log Analytics agent, Azure Platform Logs, Event Grid)
- manage Access control to the monitoring platform

## 2 Develop a Site Reliability Engineering (SRE) strategy

### 2.1 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)

### 2.2 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
- leverage Application Insights Smart Detection and Dynamic thresholds in Azure Monitor

### 2.3 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
- design approach for piecemeal recovery (e.g., to improve recovery time objective strategies)
- integrate health check with compute environment
- implement different types of health checks (container liveness, startup, shutdown)

## 3 Develop a security and compliance plan

### 3.1 Design an authentication and authorization strategy

- design an access solution (Azure AD Privileged Identity Management (PIM), Azure AD Conditional Access, MFA, Azure AD B2B, etc.)
- implement Service Principals and Managed Identity
- design an application access solution using Azure AD B2C
- configure service connections

### 3.2 Design a sensitive information management strategy

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

### 3.3 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 Code scanning, GitHub Secret scanning, pipeline-based scans, Git hooks, SonarQube, Dependabot, etc.)

### 3.4 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
- design break-the-glass strategy for responding to security incidents

## 4 Manage source control

### 4.1 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
- design approach for efficient code reviews (e.g., GitHub code review assignments, schedule reminders, Pull Analytics)

### 4.2 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

### 4.3 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

### 4.4 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)
- integrate GitHub Codespaces

## 5 Facilitate communication and collaboration

### 5.1 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, ServiceNow)
- integrate GitHub as repository with Azure Boards
- communicate user analytics

### 5.2 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)

### 5.3 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 and project management platforms (e.g., Email, SMS, Slack, Teams, ServiceNow, etc.)
- integrate build and release with communication platforms (e.g., build fails, release fails)
- integrate GitHub pull request approvals via mobile apps

# 6 DEFINE AND IMPLEMENT CONTINUOUS INTEGRATION

## 6.1 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)

## 6.2 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

## 6.3 DESIGN AN APPLICATION INFRASTRUCTURE MANAGEMENT STRATEGY

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

## 6.4 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)

### 6.5 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

### 6.6 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.)

## 7 DEFINE AND IMPLEMENT A CONTINUOUS DELIVERY AND RELEASE MANAGEMENT STRATEGY

### 7.1 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.)

### 7.2 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, Azure App Configuration)
- design and implement release gates and approval processes

### 7.3 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