AZ-400 Azure DevOps Engineer

Overview

This seven-MOC packaged set aligned to Azure Exam: Azure DevOps Engineer contains courseware that helps prepare students for Exams AZ-400. Passing this exam is required to earn the Azure DevOps Engineer certification.

Prerequisite Comments

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.

Target Audience

Students in this course are interested in implementing DevOps processes or in passing the Microsoft Azure DevOps Solutions certification exam.

Course Objectives

After completing this course, students will be able to:
- Describe the benefits of using source control
- Migrate from TFVC to Git
- Scale Git for Enterprise DevOps
- Implement and manage build infrastructure
- Manage application config & secrets
- Implement a mobile DevOps strategy

Course Outline

Getting started with Source Control

What is Source Control?
Benefits of Source Control
Types of source control systems
Introduction to Azure Repos
Migrating from TFVC to Git
Authenticating to your Git Repos
Scaling git for enterprise DevOps
How to structure your git repo? Mono Repo or Multi-Repo?
Git Branching workflows
Collaborating with Pull Requests
Why care about GitHooks?
Fostering Internal Open Source
Git Version
public projects
Storing Large files in Git

Implement & Manage Build Infrastructure
The concept of pipelines in DevOps
Azure Pipelines
Evaluate use of Hosted vs Private Agents
Agent pools
Pipelines & Concurrency
Azure DevOps loves Open Source projects
Azure Pipelines YAML vs Visual Designer
Setup private agents
Integrate Jenkins with Azure Pipelines
Integration external source control with Azure Pipelines
Analyze & Integrate Docker multi stage builds

Managing application config & secrets
Demo: SQL Injection attack
Implement secure & compliant development process
Rethinking application config data
Manage secrets, tokens & certificates
Implement tools for managing security and compliance in a pipeline

Implement a mobile DevOps strategy
Introduction to Visual Studio App Center
Manage mobile target device sets and distribution groups
Manage target UI test device sets
Provision tester devices for deployment
Provision tester devices for deployment

Implementing Continuous Integration in an Azure DevOps Pipeline
Continuous Integration Overview
Implementing a Build Strategy

Managing Code Quality and Security Policies
Managing Code Quality
Managing Security Policies
Implementing a Container Build Strategy

Design a Release Strategy

Introduction to Continuous Delivery
Release strategy recommendations
Building a High Quality Release pipeline
Choosing a deployment pattern
Choosing the right release management tool
Building a release strategy
Differentiate between a release and a deployment
Define the components of a release pipeline
Explain things to consider when designing your release strategy
Classify a release versus a release process, and outline how to control the quality of both
Describe the principle of release gates and how to deal with release notes and documentation
Explain deployment patterns, both in the traditional sense and in the modern sense
Choose a release management tool

Set up a Release Management Workflow

Introduction
Create a Release Pipeline
Provision and Configure Environments
Manage And Modularize Tasks and Templates
Integrate Secrets with the release pipeline
Configure Automated Integration and Functional Test Automation
Automate Inspection of Health
Building a release management workflow
Explain the terminology used in Azure DevOps and other Release Management Tooling
Describe what a Build and Release task is, what it can do, and some available deployment tasks
Classify an Agent, Agent Queue and Agent Pool
Explain why you sometimes need multiple release jobs in one release pipeline
Differentiate between multi-agent and multi-configuration release job
Use release variables and stage variables in your release pipeline
Deploy to an environment securely, using a service connection
Embed testing in the pipeline
List the different ways to inspect the health of your pipeline and release by using, alerts, service hooks and reports
Create a release gate
Implement an appropriate deployment pattern

Introduction into Deployment Patterns
Implement Blue Green Deployment
Implement Canary Release
Implement Progressive Exposure Deployment
Describe deployment patterns
Implement Blue Green Deployment
Implement Canary Release
Implement Progressive Exposure Deployment

Designing a Dependency Management Strategy

Introduction
Packaging dependencies
Package management
Implement versioning strategy
Recommend artifact management tools and practices
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

Manage security and compliance

Introduction
Package security
Open source software
Integrating license and vulnerability scans
Inspect open source software packages for security and license compliance to align with corporate standards
Configure build pipeline to access package security and license rating
Configure secure access to package feeds

Infrastructure and Configuration Azure Tools

Learning Objectives
Infrastructure as Code and Configuration Management
Create Azure Resources using ARM Templates
Create Azure Resources using Azure CLI
Create Azure Resources by using Azure PowerShell
Additional Automation Tools
Version Control
Azure Deployment Models and Services

Learning Objectives
Deployment Models and Options
Azure Infrastructure-as-a-Service (IaaS) Services
Azure Automation with DevOps
Desired State Configuration (DSC)
Azure Platform-as-a-Service (PaaS) services
Azure Service Fabric

Create and Manage Kubernetes Service Infrastructure

Learning Objectives
Azure Kubernetes Service

Third Party and Open Source Tools available with Azure

Learning Objectives
Chef
Puppet
Ansible
Cloud-Init
Terraform

Implement Compliance and Security in your Infrastructure Lessons

Security and Compliance Principles with DevOps
Azure Security Center

Recommend and design system feedback mechanisms

The inner loop
Continuous Experimentation mindset
Design practices to measure end-user satisfaction
Design processes to capture and analyze user feedback from external sources
Design process to automate application analytics

Implement process for routing system feedback to development teams

Implement tools to track system usage, feature usage, and flow
Implement routing for mobile application crash report data
Develop monitoring and status dashboards
Integrate and configure ticketing systems with development team's work management system
Optimize feedback mechanisms

Site Reliability Engineering
Analyze telemetry to establish a baseline
Perform ongoing tuning to reduce meaningless or non-actionable alerts
Analyze alerts to establish a baseline
Blameless PostMortems and a Just Culture

Planning for DevOps

Transformation Planning
Project Selection
Team Structures

Planning for Quality and Security

Planning a Quality Strategy
Planning Secure Development

Migrating and Consolidating Artifacts and Tools

Migrating and Consolidating Artifacts
Migrating and Integrating Source Control