**ADO**

ADO Branching Strategies:

* Trunk-based development:
  + All development work takes place on the main branch (the "master")
  + Emphasizes continuous integration, with devs committing small, incremental changes to the main branch
  + Maintains code quality and stability
* Feature based development:
  + Encapsulates each new feature or change in a dedicated branch separate from the main branch (derived from feature branches)
  + Develop and test new features in isolation, and merge to master once it's complete and tested
  + Allows for parallel development, facilitates code review, and ensures main branch remains stable
* Release based development:
  + Dedicated branch is created for a release, with coding and testing for the release occurring there
  + Once the release is considered ready, it's merged back to the master branch
  + Allows for a controlled and organized release process
* Forking workflow:
  + Instead of using a single server-side repository as the central codebase, it gives every developer a server-side repository
  + Each contributor has two Git repositories: a private one and a public server-side one
  + Promotes a high degree of parallel development and collaboration
* GitHub Flow is a popular trunk-based development release flow, which stipulates that pull requests must deploy to production for testing before they can merge to the main branch. This process means that all pull requests wait in the deployment queue for the merge.
  + In the GitFlow workflow strategy, the main branch represents the source of the code that is in production. The develop branch serves as an integration branch for features.

ADO Policy types:

* Branch policy can be configured to enforce various requirements for pull requests. Branch policies minimizes administrative effort by automating the enforcement of branch requirements
  + Check for linked work items
  + Check for comment resolution
  + Require a minimum number of viewers
  + Status checks (other services via API)
* Build policies focus on ensuring committed code compiles successfully or automated tests pass before merging changes
* Status policies enforce that certain conditions are met based on the status posted to the pull request, such as enforcing checks and status from external services
* Check-in policies apply to Team Foundation Version Control (TFVC)

Types of identities/access:

* Managed Identity (system-assigned and user-assigned): **For Azure resources accessing other Azure services**, eliminating credential management
  + User-assigned managed identity = allows for creating a single identity that can be used by multiple Azure resources, including virtual machines. Lifecycle independent from resources.
  + System-assigned managed identity = tied to a specific Azure resource (like a virtual machine) and is automatically created and deleted with that resource. Tied to resource lifecycle.
* Service principal = can be used to represent an application or service's identity, but does not automatically manage credentials like a managed identity. Can be used to create and manage resources such as load balancers and managed disks in Azure. **For automated deployments, CI/CD pipelines, and application authentication to Azure resources**
* Personal Access Tokens (PATs): **For accessing Azure DevOps REST APIs, Git repositories, and third-party integrations**
* Shared Access Services (SAS) tokens are primarily **used in Azure Storage services** to grant limited access to objects in the storage account

Authentication methods and use cases:

* Azure Active Directory (Azure AD) Integration: Single sign-on, user authentication, conditional access policies
* SSH Keys: Git repository access, secure remote connections
* OAuth 2.0 and OpenID Connect: Third-party application integration, delegated access scenarios
* Certificate-Based Authentication: Service-to-service authentication, high-security environments
* Token-Based Authentication (JWT): Header, payload, signature

Metrics:

* Cycle time = measures the amount of time it takes for a work item to go from the start of work to completion
* Lead time = measures the time from when the work item was created until it is closed, which includes the time before work actually starts on the item. (faster outcome metric)
* Velocity = the amount of work a team completes during a sprint or iteration
* Burndown chart = used to track the amount of work remaining against the time available, typically in a sprint or release
* Burnup chart = tracks the total amount of work done over time
* Defect escape rate = measures the number of defects discovered in production compared to the number found during the development phase.
* Unplanned work rate = measures the amount of work that arises unexpectedly, which was not initially planned or part of the project scope.
* Defect rate = refers to the frequency of defects in the development process or product
* Rework rate = measures the amount of work that needs to be redone or corrected, which is indicative of the quality of work.
* Deployment speed = measures how quickly new features are delivered and deployed to a live environment, from time that code is committed to time that it's up and running
* CFD (Cumulative Flow Diagram) widget: monitor and count work items as they move to a different state
* Mean Time to Recover = quality and security metric
* Server to Admin Ratio = efficiency metric
* Application Performance = efficiency metric

ADO Gates:

* Pre-deployment gates ensure there are no active issues in the work item or problem management system before deploying a build to an environment.
* Post-deployment gates ensure there are no incidents from the monitoring or incident management system for the app after it’s been deployed, before promoting the release to the next environment.
* Flaky tests = allow you to identify and handle tests that might fail intermittently

Semantic versioning (SemVer) = x.y.z

* X = breaking changes
* Y = new features, but backwards compatible
* Z = bugs
* A patch element is the only answer that provides assurances of backward compatibility

Calendar versioning (CalVer) = x.y.z

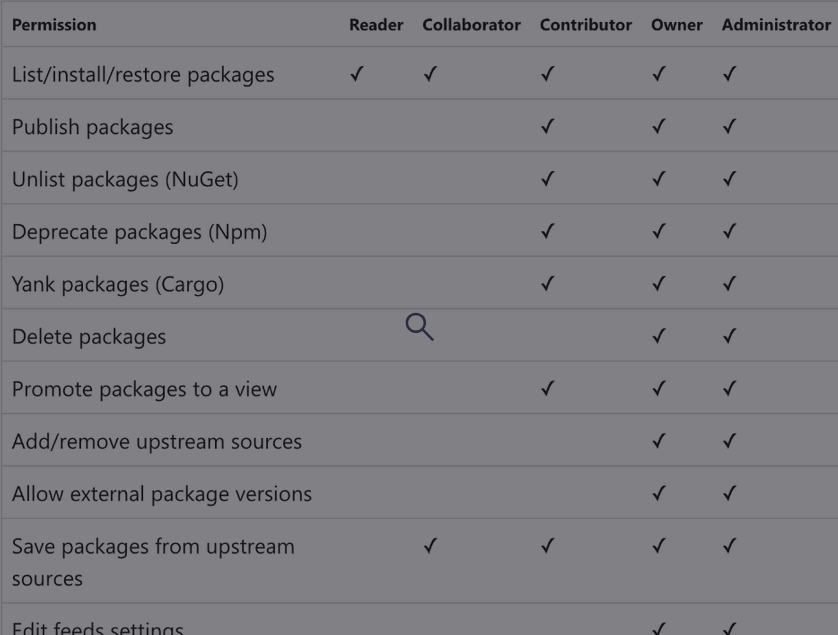
* X = year or major time component
* Y = sub-time component such as month or quarter
* Z = day or patch level identification

ADO Boards/Working with ADO tickets:

* To assign dependencies to tickets in ADO: Add link, then add as Successor
* To link a GitHub commit to a Azure Boards work item, include a specific syntax in your commit title or description. (AB# followed by the work item number).
  + Resolve work items on commit with "fix", "fixed", or "fixes"
  + Custom fields can be used to track additional info for a dashboard

ADO Project roles:

* The Reader role in Azure DevOps is primarily focused on providing read-only access to project resources. Users in this role can view information, work items, and other project details. However, this role does not grant individuals to create or modify resources, including custom work item queries.
* Collaborator = read + save packages from upstream sources
* The Contributor role is designed for users who actively contribute to a project's progress, it has a set of permissions that includes creating, reading, updating, and deleting resources within a specified project.
* Project administrators provide significant control over a specific project, it includes the ability to create and modify work items, create repositories and manage permissions
* The Project Collection Administrators group has a significant administrative control over the organizations and multiple projects.
* Build Administrators = intended for users who manage build pipelines and services.



ADO GitHub repository roles:

* Read = to view the project
* Triage = for contributors who need to manages issues/discussions/PR without Write access
* Write role = allows users to actively push changes to the repository
* Maintain = for PMs who need to manage the repo without access to sensitive or destructive actions
* Admin = full access to the project, including managing security and destroying a repo

ADO Agent Pools:

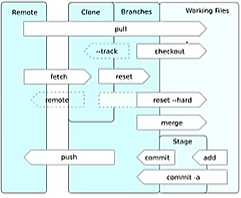
* Agent pools consist of self-hosted and Microsoft-hosted agents responsible for executing tasks and jobs during your build and release processes.
* TaskAgentPoolSizeSnapshots offers a window into the performance, availability, and capacity of these agent pools over time.
* The pool consumption report at the organization level provides a comprehensive view of how your agent pool is utilized throughout the organization.
* Microsoft-hosted agent pools cannot access on-premises systems such as dependency management - a self-hosted agent would be needed
* Self-hosted build agents = startup time is reduced because they can be pre-configured with necessary tools and dependencies.
  + - Self-hosted agents can push things like build outputs to ADO and source code into a Git repo
    - Only self-hosted agents support incremental builds
* Purchasing more parallel jobs for a pipeline extends the maximum allowed job duration from 60 minutes to 360 minutes
* Reviewing the historical graph for agent pools allows you to visually analyze the utilization of agents over time
* Projects that use public repositories are eligible for free parallel jobs in Azure Pipelines
* Long build times and frequently queued builds indicate a need for parallel jobs
* Maximum build time for a Microsoft-hosted agent is 6 hours

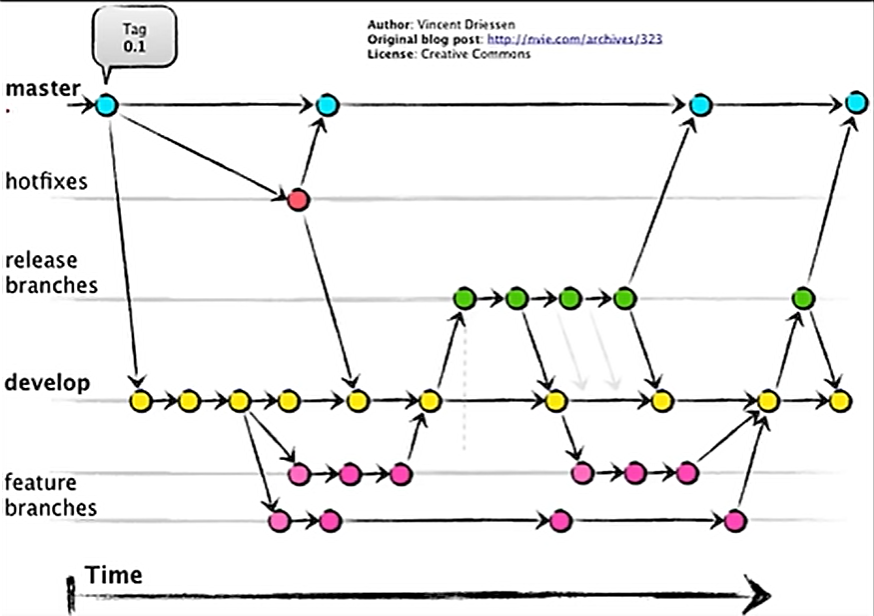
**Git/GitHub**

Git commands:

* bfg = BFG Repo-Cleaner is a tool specifically designed for cleaning unwanted data from Git repositories, particularly large files or sensitive data committed accidentally (alternative to git-filter-branch) Format = "bfg --delete-files <filename>"
  + git add = stages files for commit
  + git checkout = switch to a branch (or create a new one)
  + Git cherry-pick = apply specific commits from one branch to another without merging the entire branch. (common for prod hotfixes and security patches)
  + git clean = deletes untracked files but does not undo a commit.
  + git clone = Creates a complete local copy of a repository, including its history.
    - git clone –-filter=blob:none - specifies that blobs (file content) should not be downloaded during the initial clone (You can still search the commit history)
    - git clone –-depth-1 - only fetches the latest history of the repo and only includes the latest commit in history
    - git clone --bare = clones with folder name including the .git suffix
  + git commit = Represents a change of data in the local repository. commit staged changes
  + git fetch = Downloads data from a remote repo without integrating it into your work.
  + git filter-branch = allows for rewriting the Git history of branches, which can be used to remove files from the entire history of the repository.
  + git filter-repo = for rewriting the history of a git repo.
  + git gc = garbage collection. --aggressive means a more forceful garbage collection
  + Git LFS = handles large files by storing in the repository only pointers to them, rather than the actual files. When you clone the repository, the pointer file is used to locate the large file. Git LFS supports the git lfs clone command that offers superior performance as compared to the traditional git clone command.
  + Git merge = Combines multiple commit histories into one. Integrates commits from a source branch into your current local branch (target branch)
    - Fast-forward merge - occurs when the target branch has not diverged from the source branch - there's a direct linear path from the current branch tip to the merge target. Git simply moves the branch pointer forward to the latest commit, no new merge commit is created. (git merge [feature-branch])
    - No-fast-forward merge (also a three-way merge) = always creates a new merge commit, even when a fast-forward would be possible, explicitly showing the merge operation in history. Creates a new merge commit with two parent commits (git merge –no-ff [feature branch])
    - Rebase = like a fast-forward merge, but onto a new branch. Replaces the commit history of one branch with another (git rebase)
    - Squash merging (git merge --squash [feature-branch]) = where all the changes in a feature branch are combined into a single commit when merged into master branch. Greatly reduces the number of individual commits added to the master branch and reduces volume of history.
  + git pull = downloads changes from a remote repository and merges them into your branch
  + git push origin = Uploads your local repository changes to a remote repository
    - git push --mirror = overwrites all branches in the target repo, including deleting branches that aren't in the source repo
  + git rebase = another type of merge. It integrates source branch commits into the target branch, it does not undo changes.
  + Git reflog = Reference logs, or "reflogs", record when the tips of branches and other references were updated in the local repository. Can be used to look at old log entries or delete items from the reflog.
  + Git reset = undoes local changes, with options to unstage or revert commits.
    - git reset --mixed HEAD undoes the staging of files but keeps file changes. It does not affect commits.
  + git revert = allows you to undo a change in a shared branch.
  + git rm = deletes the file from the working directory and stages the file's removal in a new commit
* git sparse-checkout = creates a sparse checkout, which changes the working tree from having all tracked files present to having only a subset of those files
* git stash = allows you to temporarily store your current work, for later use
  + git submodule add = adds a repo as a submodule of another repo for dependencies and code reuse. Format = "git submodule add <url>"
* git tag = used to label releases for easy reference. There are two types of tags: lightweight and annotated
  + git tag -a = annotated - containing additional information like the tagger's name, date, and a message.
* scalar clone = a command designed for efficient cloning of large repositories. It uses a technique called sparse checkout, which downloads only the files necessary for the current branch and working directory. As a result, it significantly reduces the time and storage space required for cloning extensive repositories. Format = "scalar clone <url>"

Git visuals:





Git terms:

* Repository: Represents the logical container holding the Codebase
* Tree: Represent the entire history of a repo
* Remote: A version of your project hosted elsewhere, used for exchanging commits.
* Branches: Divergent paths of development, allowing isolated changes.
* Main (formally known as master) the most common name for the default branch
  + Modern terms = develop = "main", master = "prod", release branches = "staging"

Basic dev flow:

* git add .
* git commit
* git push origin

To recover a deleted branch:

* git log - This is to find the SHA1 of the last commit on the deleted branch.
* git checkout <SHA1> - This is to switch to the state of the repository that the commit hash represents.
* git branch [new branch name] - This is to create a new branch named [new branch name] starting from the current commit you have checked out (the last commit of the deleted branch).

Git repo features:

* Root directory of a Git repo = "repo1/src"
* gitHub URL format:<https://github.com/RepositoryOwner/RepositoryName/>
* The **CODEOWNERS** file specifies who is responsible for different sections of a codebase. When a pull request (PR) is opened that touches files listed in the CODEOWNERS file, the designated code owner is automatically asked to review that section of the code
* gist is used as a smaller version of github repo (individual file or collection of files) for sharing and code collaborations for code snippets or small projects.
* Github Enterprise is an on-premises version of Github that allows you to host Git repositories on your own infrastructure
* GitHub secrets can be created at the organization and repository levels. You can use secrets in a step of an action within a workflow, but you cannot create them at any of the other three levels.

**Third-party tools:**

Testing tools/frameworks

* GitHub Advanced Security:
  + Code Scanning (CodeQL) - Static Application Security Testing (SAST) for identifying security vulnerabilities in source code
  + Secret Scanning - Detect and prevent secrets (API keys, tokens, passwords) from being committed to repositories
  + Dependency Review (Dependabot) - Automated dependency management and vulnerability scanning for project dependencies
* Playwright = a framework specifically designed for automating UI testing across various web browsers.
* JaCoco = a code coverage library for Java application
* Xamarin.UITest = primarily used for testing mobile applications built with Xamarin.
* Microsoft.CodeAnalysis = a set of .NET Compiler Platform ("Roslyn") APIs for code analysis
* Pester = a PowerShell testing framework that can be used for both unit testing and integration testing
* Mend Bolt (WhiteSource Bolt) = designed for managing open source security, scanning project dependencies for open source vulnerabilities and licensing issues
* Black Duck = manages the risk with open source software, such as licensing compliance, security vulnerabilities, and operational risk
* Code Climate = primarily known for its code analysis duplication, automated code review, analytics and also it covers various aspects of code quality. It mainly focuses on overall code health.
* Rollbar = an error tracking and monitoring tool designed to help developers in identifying and resolving errors in the applications. It helps in providing real-time errors, grouping similar issues, and identifying the errors at specific lines of code
* DeepSource = a valuable tool for static code analysis and also helps developers by improving the code health, code quality, code security, and efficiency in code reviews.
* SonarCloud = a cloud-based static code analysis platform that provides continuous code quality and security analysis for DevOps pipelines
* Maven = a build automation tool used primarily for Java projects. It can integrate with SonarQube to perform code analysis and quality checks as part of the build process
* Gradle = a build automation tool used primarily for Java projects.

Powershell tools

* + Powershell DSC = automatically configures VMs to a desired state (uses MOF files)
* PSScriptAnalyzer = used for analyzing PowerShell scripts to ensure they follow best practices and coding standards
* PSCodeHealth = a tool for assessing the health and quality of PowerShell code by generating metrics and insights
* Test-DscConfiguration = used to verify if the current environment matches the desired state configuration

Powershell commands:

* GetAzResource - ResourceType [Azure resource] getting properties of a specific type of resource

Monitoring Tools:

* OpenTelemetry = allows creation and management of telemetry data in Application Insights
* OpenCensus = an OpenTelemetry predecessor that monitors and traces web servers. Also works with Application Insights

Tools for large repos:

* bfg = BFG Repo-Cleaner is a tool specifically designed for cleaning unwanted data from Git repositories, particularly large files or sensitive data committed accidentally (alternative to git-filter-branch). The BFG utility can quickly delete a subset of repository files based on criteria such as size, as well as automatically update all commits and tags.
* Git Large File Storage (LFS) is used for handling large files by storing references to them in the repository, rather than the files themselves
* Scalar = Microsoft's tool designed to optimize Git performance for very large repositories, particularly monorepos used in enterprise environments.
  + Downloads only essential Git objects initially (with git sparse-checkout)
  + Fetches additional objects on-demand when needed
  + Significantly reduces initial clone time and disk usage
  + Particularly effective for repositories with extensive history

Comparison of source control systems:

* Git = dominates modern development due to its flexibility and distributed nature.
* Perforce = excels in enterprise environments with large binary assets.
* TFVC = works well for Microsoft-centric organizations.
* SVN = remains viable for teams preferring centralized, straightforward version control but is generally being replaced by Git in most contexts.

Other tools:

* Microsoft Teams for GitHub app = designed to integrate GitHub with Microsoft Teams directly and can be used to create subscriptions for various GitHub events, including commits.
* Microsoft Defender for Containers = provides the capability to scan container images for vulnerabilities within the Azure DevOps pipeline, ensuring that the images are secure before being deployed to a Kubernetes cluster
* Github Codespaces is a cloud-based development environment that fully integrates with GitHub and supports a range of devices, including browsers, tablets, and Chromebooks. It offers integrated debugging tools and is ideal for remote workers and hot-desking environments, as it allows developers to access their development environment from anywhere.
* Dependency Tracker extension = specifically designed for tracking and managing dependencies across multiple projects in Azure DevOps. It allows you to visualize and manage the relationships between different work items, offering a clear overview of consumers and producers for each project.
  + Link width = number of dependencies
  + Link color = risk level of dependency
  + Link length = distance between two nodes
  + Node color = status or category of the component
* GitHub Importer = simplifies the process of importing repositories from other VCS's to GitHub
* IT Service Manager Connector (ITSM) - ensures that work items, like incidents and change requests, are created based on alerts and log data from Azure Monitor and Log Analytics
* Mermaid is a text-based diagramming tool that integrates seamlessly with Markdown, which is ideal for use in GitHub repositories. It allows for storing diagrams as plain text, thereby simplifying version control, modification, merging, and reuse.
* Azure Artifacts Credential Provider = automates the authentication process when accessing authenticated feeds like Azure Artifacts in Azure DevOps.
* The TFS Database Import Service = the most direct and comprehensive method provided by Microsoft for migrating from TFS to Azure DevOps.

**Azure Products:**

Azure Application Insights:

* Specialized for application-level metrics and the telemetry of web applications
* Features:
  + - Profiler in Azure Application Insights is used to track performance issues, allowing you to inspect execution profiles of sampled requests.
    - Performance tracking - Request response times and throughput, server performance counters (CPU, Memory, Disk)
    - Application Map in Azure Application Insights provides a visual topology of the application components, not functionality for alerting on status codes returned by the front end.
    - Metrics explorer provides direct access to aggregated data, including rates of requests, failures, and exceptions.
    - Usage analysis allows you to analyze user segmentation and retention.
    - Smart Detection = automatic anomaly, failure rate spike, and memory leak detection
    - Synthetic Monitoring = availability tests from global locations (URL, ping, multi-step web tests)
* Telemetry sampling rate for Application Insights must be configured in the app startup code
  + - Appsettings.json = the file that we need to modify the “Applicationinsights” section and ADD the application insights’ instrumentation key or a connection string over there.
  + Provides direct integration with Azure Boards, allowing you to embed its data into work items (proactive detection and individual instances of activity)
* KQL key commands:
* where = filters a query
* Summarize = group data points to display
* Bin(TimeGenerated, 1h) = lets you group time-based parameters together in the duration specified
* Extend = operator that lets you introduce a new column in a result set without creating a separate index (extend the results with additional columns)
* Project = list which fields will be presented for output (data points to display)
* Render = visualizes the results of the query as a chart ("render timechart" = time chart visualization)

KQL sample query:

AzureDiagnostics

| where Category = "DscNodeStatus"

| where OperationName contains "DscNodeStatusData"

| where ResultType != "Compliant"

[In sample tests, "where" clauses come before "join" clauses]

Azure App Service:

* A fully managed platform as a service (PaaS), to create web applications in popular languages like .NET, Java, Node.js, and Python.
* Includes provisions for implementing staging environments and deploying apps
* Diagnostics:
  + - The Availability and Performance category is used to analyze issues related to app uptime and responsiveness
    - Navigator helps you understand the overall architecture and dependencies of the app
    - Configuration and Management is focused on configuration settings, deployments, and management operations of the app
    - Diagnostic Tools can assist with troubleshooting specific technical issues

Azure Artifacts views:

* Feed views enable developers to share a subset of package-versions with their customers
  + @default = view that contains all packages
  + @release = a view for distributing only packages ready for release (i.e. tested)
* Adding the package URL to the Nuget Package manager settings in Visual Studio allows VS to access the package from the specified feed

Azure Automation:

* Azure Automation is a process automation tool hosted in Azure. It provides the ability to run scripts or Runbooks. In addition, we can manage inventory and change tracking with Configuration Management, keep computers up to date with Update Management, and share resources such as modules, schedules, and credentials with share resources.
* offers configuration management for an application infrastructure, not only in Azure
* Provides a built-in PowerShell DSC pull server
* Configure the Source control settings for the Automation account to ensure that every committed change to the runbook code in Azure DevOps automatically updates and publishes the runbook to Azure Automation
* 500 minutes each month to run things with your Azure Automation Account
* Three core areas:
  + - Configuration Management: how do you control the desired state of your environments? Inventory, Change Tracking, Desired State Configuration
    - Update Management: where you can access and deploy patches for your hybrid envs from a single pane of glass
    - Process Automation: to orchestrate across your environments (jobs, runbooks)

Azure Automation State Configuration:

* Azure Automation State Configuration is an Azure configuration management service that allows you to write, manage, and compile PowerShell Desired State Configuration (DSC) [configurations](https://learn.microsoft.com/en-us/powershell/dsc/configurations/configurations) for nodes in any cloud or on-premises datacenter.
* can work centrally for Azure and on-premises virtual machines

Azure Functions:

* Azure Functions provides the ability to create and run custom availability tests by relying on the TrackAvailability() method (included in the Azure SDK for .NET)

Azure Key Vault:

* Access policies in Azure Key Vault are the correct place to configure permissions for identities (such as service principals or managed identities) to access the secrets, keys, and certificates in the vault.
  + Get permission: allows applications to retrieve values of secrets stored in Key Vault
  + List permission: allows applications to see what secrets are available in Key Vault
* IAM settings determine who has management access to the Key Vault resource itself within the Azure subscription, it does not grant fine-grained permissions to the data stored within the Key Vault like access policies do.
* Creating a variable group allows you to link Azure Key Vault secrets to your Azure DevOps pipelines

Azure Log Analytics:

* A central repository for collecting and analyzing telemetry data across Azure services
* Service Map in Azure Log Analytics is used to visualize complex interdependencies and connections within your application infrastructure, rather than monitoring and alerting on the availability of endpoints.
* Analytics = allows for the parsing of logs from multiple sources and the identification of the root cause of issues
* Synthetic monitoring = involves simulating user interactions with an application to test its performance and availability

Azure Monitor:

* Workbooks are used for interactive data visualization
  + Uses action groups as a collection of notification preferences when an alert is triggered. Action groups define the action to take when an alert is triggered.
* Application Insights is part of Azure Monitor

Azure Pipelines:

* The main file is called azure-pipelines.yml, it’s a trigger for Azure to know you’re creating a pipeline
* To download a public container image in Azure Pipelines, you need a service connection to a container registry where the image is stored. (Docker registry)
  + - Service connections used short-lived credentials that get created on the fly when the connection is established - you don’t have to worry about rotating credentials, etc
* Pipeline artifacts = for using specific files from a job in another job
* Pipeline caching = for improving build time by reusing files and dependencies from previous runs of the same job
* Setting the System.Debug variable to true in a pipeline will enable verbose logging, which is useful for diagnosing problems during the build or deployment process.
* There are two types of templates: includes and extends.
  + Includes templates let you insert reusable content with a template.
  + Extends templates let you control what is allowed in a pipeline. For example, an extends template can be used in the context of extending a pipeline to perform stages or jobs.
* Container jobs in Azure Pipelines run each job in a container with its own dependencies, effectively resolving issues with conflicting dependencies
* Caching is only available in YAML and classic build only pipelines, not classic release
* Pipeline run summaries:
  + - From Related, you can download coverage extension files to be used as evidence of code coverage. Tests and Coverage can be used to configure tests and coverage or to see the general results of coverage but not to generate evidence by using the Visual Studio Test task. Tests are used to see test results but not to get evidence about code coverage. Repository and version provides information about the repository used for the pipeline.
  + An OData feed provides a straightforward approach to publishing reports that show Azure Pipelines analytics
  + Multi-job configuration is used for pipeline performance optimization, allowing you to run multiple jobs on multiple agents in parallel
  + With secure files, you can store files in Azure DevOps to be shared across pipelines, as well as assign proper permissions
  + An environment represents a collection of resources targeted for deployment

Azure pipelines YAML:

yaml

*# Pipeline metadata*

name: string

trigger: trigger definition

pr: pr trigger definition

schedules: cron schedules

resources: external resources

variables: variable definitions

pool: agent pool definition

*# Pipeline execution hierarchy*

stages:

- stage: string

jobs:

- job: string

steps:

- step definition

* **Stages:** High-level phases (Build, Test, Deploy)
* **Jobs:** Units of work that run on agents
  + Can be used to run steps like tests on different machines ( pool:), or to run steps in parallel
* **Steps:** Individual tasks within jobs
  + Task: Execute predefined Azure DevOps tasks
  + Script: Execute custom shell commands
  + Checkout: Download source code or artifacts
  + Download: Download build artifacts
* - trigger: for when code is pushed to a branch
* - pr: for only when a pr is created
* succeededOrFailed() - ensures run even if the previous steps fail (but not if stages are skipped)
* always() - always ensures run
* checkout
* ApplyOnly = apply and nothing further
* ApplyAndMonitor = apply + monitor the system for drifts from the desired state and report them
* ApplyAndAutoCorrect = apply + it will automatically reapply the correct configuration

${{parameters.parameterName}} is CORRECT for referencing a parameter in a YAML template for Azure Pipelines.

Templates: for reusable code that you can call in your main azure-pipelines.yml file

* You can have templates for stages, jobs, steps
* You can have templates within templates, etc

Credential Security in pipeline:

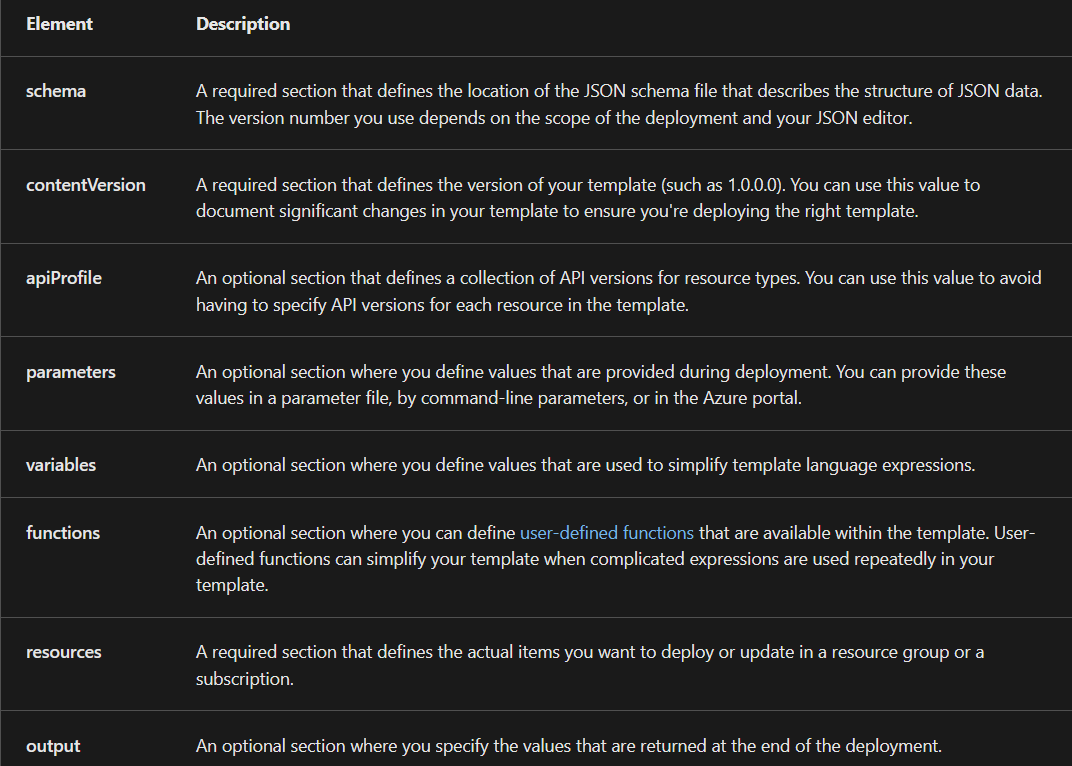
* Secret variables in a variable group ensure that credentials are not exposed in logs or scripts and can only be accessed by authorized pipelines and users.
* Storing secrets in environment variables is a secure practice in Azure Pipelines (not logged). Storing secrets in pipeline variables will expose them to the logs

Azure Repos:

* Dependency graph = identifies which components are shared across a various parts of an application and how they interact

Azure Resource Manager (ARM):

* Features of an ARM template (JSON):



* Bicep provides a simpler, more readable syntax for creating ARM templates

Azure Security Center:

* Includes a vulnerability assessment feature for VMs
* Provides a container security feature that includes vulnerability scanning for container images in Azure Container Registry

Azure Test Plans:

* A browser based test management tool within Azure Pipelines for a unified, central view of test cases, whether automatic or manual

Azure Traffic Manager:

* Types of traffic distribution methods:
  + **Priority**: a primary service endpoint for all traffic with backups if its unavailable
  + **Weighted**: to distribute traffic across a set of endpoints, either evenly or according to weights, which you define.
  + **Performance**: when you have endpoints in different geographic locations, and you want end users to use the "closest" endpoint for the lowest network latency.
  + **Geographic**: users are directed to specific endpoints (Azure, External, or Nested) based on which geographic location their DNS query originates from.
  + **Multivalue**: for profiles that can only have IPv4/IPv6 addresses as endpoints. When a query is received for this profile, all healthy endpoints are returned.
  + **Subnet**: map sets of end-user IP address ranges to a specific endpoint within a Traffic Manager profile.

Continuous deployment should include passive penetration tests as well as SSL and infrastructure scans. Nightly test runs should include infrastructure scans and active penetration tests. Continuous integration should include an Open Source Software (OSS) vulnerability scan. The integrated development environment/pull request step should include static code analysis and code reviews.

|  |
| --- |