DUMPSARENA

Microsoft Azure DevOps Solutions

Microsoft AZ-400

Version Demo

Total Demo Questions: 94

Total Premium Questions: 944

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Configure processes and communications	115
Topic 2, Design and implement source control	98
Topic 3, Design and implement build and release pipelines	399
Topic 4, Develop a security and compliance plan	156
Topic 5, Implement an instrumentation strategy	146
Topic 6, Mix Questions	8
Topic 7, Case Study 1	6
Topic 8, Case Study 2	3
Topic 9, Case Study 3	3
Topic 10, Case Study 4	5
Topic 11, Case Study 5	5
Total	944

QUESTION NO: 1

You need !0 the merge the POC branch into the default branch. The solution must meet the technical requirements. Which command should you run?

A. git push

B. git merge --allow-unrelated-histories POC

C. git rebase

D. git merge --squash

ANSWER: B

Explanation:

`git merge --allow-unrelated-histories POC` is the appropriate command when the POC branch must be merged into the currently checked-out default branch and its history has no common ancestor with that branch. Git normally refuses a merge in this situation because independent histories can indicate an unintended attempt to combine separate repositories or separately initialized projects. The `--allow-unrelated-histories` flag explicitly confirms that combining those histories is intentional, allowing Git to create the merge result. Run the command after switching to the default branch, replacing `POC` with the actual POC branch name if it differs. Any merge conflicts that result must then be resolved, staged, and committed before the integrated history can be pushed to Azure Repos. This approach retains the commits from the POC branch and records the relationship between the two histories, which is important when the requirement is to merge the branch rather than discard or rewrite its commit history. Git documents this flag as the mechanism for merging histories that do not share a common ancestor: Git merge documentation. Azure Repos supports standard Git branch and merge workflows: Microsoft Learn: Merge Git branches.

QUESTION NO: 2

You build a Node.js app named App1 by using GitHub.

You need to test App1. The solution must ensure that App1 meets the following requirements.

• Works correctly with versions 10,12, and 14 of Node.js

• Works correctly with Ubuntu versions 22.04and 20-04

• Minimizes the runtime of the test stage

What should you use?

A. jobs

B. matrices

C. environments

D. concurrency

ANSWER: B

Explanation:

GitHub Actions matrices are designed for testing the same workflow across multiple combinations of variables. A matrix can define Node.js versions such as 10, 12, and 14 and Ubuntu runner versions such as 20.04 and 22.04. GitHub Actions expands these values into the required set of jobs, allowing each Node.js and operating-system combination to be validated independently. This provides coverage for every requested compatibility scenario without requiring separate manually duplicated workflow definitions.

Matrix-generated jobs run in parallel by default when runners are available, which minimizes the elapsed runtime of the test stage compared to executing each platform and runtime-version combination sequentially. The workflow can expose matrix

values through the `matrix` context, for example to select the runner with `runs-on: ubuntu-${{ matrix.os }}` and install the matching Node.js version with the setup-node action. GitHub also supports controls such as `max-parallel` when capacity needs to be limited, while retaining matrix-based combination management. See the GitHub Actions documentation for using a matrix for jobs and Microsoft guidance on GitHub Actions in Azure DevOps practices.

QUESTION NO: 3

You have a project in Azure DevOps named Project1. Project1 contains a build pipeline named Pipe1 that builds an application named App1.

You have an agent pool named Pool1 that contains a Windows Server 2019-based self-hosted agent. Pipe1 uses Pool1.

You plan to implement another project named Project2. Project2 will have a build pipeline named Pipe2 that builds an application named App2.

App1 and App2 have conflicting dependencies.

You need to minimize the possibility that the two build pipelines will conflict with each other. The solution must minimize infrastructure costs.

What should you do?

A. Add another self-hosted agent.

B. Add a Docker Compose task to the build pipelines.

C. Change the self-hosted agent to use Red Hat Enterprise Linux (RHEL) 8.

D. Create two container jobs.

ANSWER: D

Explanation:

Create two container jobs. A container job runs the pipeline job steps inside the container image specified by that pipeline, rather than installing or relying on the application's dependencies directly on the shared self-hosted agent. Defining separate images for App1 and App2 allows each image to include its required SDK versions, libraries, tools, and environment configuration. Consequently, the build environment is isolated and reproducible for each application, substantially reducing dependency-related interference between the pipelines.

This solution also uses the existing Pool1 agent as the job host, so it avoids the ongoing infrastructure cost of adding and maintaining another VM or self-hosted agent solely to separate build dependencies. The Windows Server 2019 agent must be configured with a compatible Docker runtime and container images for the job requirements, but the pipeline-level separation is delivered by the individual container environments. Azure Pipelines supports running jobs in containers and treats the selected image as the environment in which job steps execute. Microsoft documents container-job behavior and prerequisites in Container jobs in Azure Pipelines and provides guidance for using Docker with self-hosted agents.

QUESTION NO: 4

You use WhiteSource Bolt to scan a Node.js application.

The WhiteSource Bolt scan identifies numerous libraries that have invalid licenses. The libraries are used only during development and are not part of a production deployment.

You need to ensure that WhiteSource Bolt only scans production dependencies.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Run npm install and specify the --production flag.

B. Modify the WhiteSource Bolt policy and set the action for the licenses used by the development tools to Reassign.

C. Modify the devDependencies section of the project's Package.json file.

D. Configure WhiteSource Bolt to scan the node_modules directory only.

ANSWER: A C

Explanation:

Run npm install and specify the --production flag, and modify the devDependencies section of the project's Package.json file. A Node.js project identifies deployable runtime packages in the `dependencies` section of `package.json`, while packages needed solely for local development, testing, build tooling, or similar non-runtime work belong in `devDependencies`. Correctly classifying the libraries is essential because it defines which packages are required by the deployed application.

Installing with `npm install --production` creates a production-focused dependency set by omitting development dependencies. Consequently, the installed dependency tree available to WhiteSource Bolt contains the runtime packages that are relevant to the production deployment, rather than development-only libraries that should not affect production license compliance. Together, correct dependency classification and a production installation ensure the scan reflects the intended production dependency scope.

Current npm documentation expresses this behavior through omitting the `dev` dependency type, including when the `NODE_ENV` environment variable is set to `production`. The underlying project metadata distinction remains the same: runtime packages are listed in `dependencies`, and development-only packages are listed in `devDependencies`. See npm install documentation and npm package.json devDependencies documentation.

QUESTION NO: 5

Your company makes use of Azure SQL Database Intelligent Insights and Azure Application Insights for monitoring purposes.

You have been tasked with analyzing the monitoring using ad-hoc queries. You need to utilize the correct query language.

Solution: You use the Contextual Query Language (CQL).

Does the solution meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. Ad-hoc analysis of telemetry and monitoring data in Azure Application Insights uses Kusto Query Language (KQL), through Azure Monitor Logs. KQL provides the query syntax for interactively filtering, aggregating, correlating, and visualizing telemetry such as requests, dependencies, exceptions, traces, custom events, and availability results. Application Insights data is available in Azure Monitor Logs, either in Application Insights tables in a Log Analytics workspace for workspace-based resources or through the Application Insights Logs experience; both use KQL for log queries.

Azure SQL Database Intelligent Insights identifies performance issues and offers diagnostic information, while related monitoring and diagnostic data can be investigated through Azure Monitor and Log Analytics using KQL. Consequently, Contextual Query Language (CQL) is not the appropriate language for the stated Azure monitoring-query requirement. The appropriate query language is Kusto Query Language. Microsoft documents that Azure Monitor log queries are written in KQL and provides Application Insights query guidance based on the same language. See Azure Monitor log query overview and Application Insights analytics.

QUESTION NO: 6

You have an Azure subscription that contains an Azure Active Directory (Azure AD) tenant.

You are configuring a build pipeline in Azure Pipelines that will include a task named Task1. Task1 will authenticate by using an Azure AD service principal.

Which three values should you configure for Task1? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. the object ID

B. the tenant ID

C. the app ID

D. the client secret

E. the subscription ID

ANSWER: B C D

Explanation:

Service-principal authentication is an application-to-application sign-in flow. The tenant ID is required because it identifies the Microsoft Entra ID directory in which Azure Pipelines must request an access token for the service principal. The app ID identifies the application registration, and is also commonly called the client ID. It tells Microsoft Entra ID which workload identity is attempting to authenticate. The client secret is the credential associated with that application identity and proves that the pipeline is authorized to use the service principal. Together, these values provide the standard client-credentials inputs: tenant, client identifier, and client credential. Azure Pipelines can use these values through an Azure Resource Manager service connection or a task configuration that authenticates as a service principal. Microsoft's Azure CLI guidance shows the same required service-principal sign-in inputs: a username/client ID, password/client secret, and tenant. Azure DevOps guidance likewise documents service-principal authentication when connecting pipelines to Azure resources. See Authenticate to Azure using a service principal and Connect to Azure from Azure Pipelines.

QUESTION NO: 7

You are building an ASP.NET Core application.

You plan to create an application utilization baseline by capturing telemetry data.

You need to add code to the application to capture the telemetry data. The solution must minimize the costs of storing the telemetry data.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Add the 99 parameter to the ApplicationInsights.config file.

B. From the code of the application, enable adaptive sampling.

C. From the code of the application, add Azure Application Insights telemetry.

D. Add the 5 parameter to the ApplicationInsights.config file.

E. From the code of the application, disable adaptive sampling.

ANSWER: B C

Explanation:

From the code of the application, add Azure Application Insights telemetry is correct because Application Insights is the Azure Monitor application-performance monitoring capability used to collect telemetry from an ASP.NET Core workload.

After the Application Insights SDK is installed and configured in the application, it can automatically capture incoming requests, outgoing dependencies, exceptions, performance counters or runtime metrics where applicable, and other telemetry needed to establish a representative utilization baseline. Custom events and metrics can also be sent when the baseline requires application-specific measurements.

From the code of the application, enable adaptive sampling is correct because Application Insights sampling reduces the volume of telemetry sent for ingestion while retaining a statistically useful representation of application behavior. Adaptive sampling dynamically adjusts its sampling rate according to telemetry volume, helping prevent excessive ingestion during high traffic periods. Since Application Insights costs are affected by the amount of telemetry ingested and retained, reducing unnecessary telemetry volume directly supports the requirement to minimize storage-related monitoring costs without eliminating baseline data collection. The ASP.NET Core SDK supports configuring Application Insights and its telemetry processing pipeline in application code. See Application Insights for ASP.NET Core applications and Sampling in Application Insights.

QUESTION NO: 8

Your company uses Azure DevOps for the build pipelines and deployment pipelines of Java based projects. You need to recommend a strategy for managing technical debt.

Which two actions should you include in the recommendation? Each correct answer presents part of the solution

NOTE: Each correct selection is worth one point.

A. Integrate Azure DevOps and SonarQube.

B. Integrates Azure DevelOPs and Azure DevTest Labs.

C. Configure post-deployment approvals in the deployment pipeline.

D. Configure pre-deployment approvals in the deployment pipeline.

ANSWER: A C

Explanation:

Integrate Azure DevOps and SonarQube. is correct because SonarQube performs continuous static analysis of Java code as part of the pipeline. Its quality model identifies maintainability problems such as code smells, duplicated code, insufficient test coverage, and security or reliability issues. These findings make technical debt visible and measurable, allowing teams to prioritize remediation as part of their normal backlog and delivery process. Azure Pipelines provides SonarQube analysis tasks that prepare the analysis, run it with the build, and publish the quality-gate result.

Configure post-deployment approvals in the deployment pipeline. is also appropriate because it introduces a controlled review point after a deployment. Stakeholders can assess the deployed application, including operational observations and feedback that can reveal maintainability or quality concerns requiring follow-up work. Recording and acting on those findings supports an ongoing technical-debt management practice rather than treating code-quality analysis as a one-time activity. Azure DevOps environments support approval checks to control progression and provide governance around deployment stages. See SonarQube analysis tasks for Azure Pipelines and Approvals and checks in Azure Pipelines.

QUESTION NO: 9

You use Azure Pipelines pipeline to build and deploy an app named App1.

You need to ensure that before App1 is deployed, all the code for the app passes a security validation by using a custom tool. What should you do?

A. Add a service hook to the project.

B. Add a status check to the policies of the branch used by your company ' s development department.

C. Add a status check to the policies of the main branch.

D. Limit the job authorization scope to the current project for all the release pipelines.

ANSWER: C

Explanation:

Add a status check to the policies of the main branch. A branch policy status check provides a merge-control mechanism for the branch from which the deployable application code is produced. The custom security tool can run during validation and publish a commit status to Azure Repos indicating whether its required security validation succeeded. Configuring that status as required on the main branch means pull requests cannot be completed into that branch until the custom validation reports success. Consequently, the code integrated into the main branch—the code normally selected by the build and deployment pipeline—has passed the security check before it can proceed through the normal deployment workflow.

Status policies are designed to require external services or tools to post a successful status for a pull request's latest commit. The status is associated with a named context, allowing the branch policy to identify the specific custom security validation that must complete. For the protection to apply consistently to the version being deployed, the policy belongs on the main integration branch rather than an unrelated development branch. See Pull request status checks and Branch policies and settings.

QUESTION NO: 10

You have an on-premises app named App1 that accesses Azure resources by using credentials stored in a configuration file.

You plan to upgrade App1 to use an Azure service principal.

What is required for App1 to programmatically sign in to Azure Active Directory (Azure AD)?

A. the application ID, a client secret, and the object ID

B. a client secret, the object ID, and the tenant ID

C. the application ID, a client secret, and the tenant ID

D. the application ID, a client secret, and the subscription ID

ANSWER: C

Explanation:

The application ID, a client secret, and the tenant ID are required to authenticate an on-premises application by using a service principal and the OAuth 2.0 client credentials flow. The application ID, now commonly called the client ID, identifies the app registration that represents App1 in Microsoft Entra ID (formerly Azure AD). The client secret is an application credential that lets App1 prove that it is the application associated with that client ID. In production, a certificate can be used instead of a client secret, but a client secret is the credential specified here.

The tenant ID identifies the Microsoft Entra tenant that issues the access token. App1 sends these values to the tenant-specific identity platform endpoint to obtain a token. It can then present that token when accessing Azure resources, provided the corresponding service principal has the necessary Azure role assignments or application permissions. This is the standard configuration used by Azure SDK client-secret authentication, including `ClientSecretCredential`, which takes tenant ID, client ID, and client secret values. For more information, see Application and service principal objects in Microsoft Entra ID and ClientSecretCredential Class.

QUESTION NO: 11

Note: The question is included in a number of questions that depicts the identical set-up. However, every question has a distinctive result. Establish if the solution satisfies the requirements.

You run the Register-AzureRmAutomationDscNode command in your company's environment.

You need to make sure that your company's test servers remain correctly configured, regardless of configuration drift.

Solution: You set the -ConfigurationMode parameter to ApplyOnly.

Does the solution meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. The goal requires continuous configuration enforcement: after a server has received its desired state configuration, any later deviation from that configuration must be detected and remediated automatically. In PowerShell Desired State Configuration, the Local Configuration Manager uses `ConfigurationMode` to determine how a node responds when its actual state differs from its assigned configuration.

The appropriate behavior for this requirement is `ApplyAndAutoCorrect`. This mode applies the desired configuration and, during subsequent consistency checks, reapplies it if configuration drift is found. Consequently, it keeps managed test servers aligned with the configuration published through Azure Automation State Configuration. Selecting `ApplyOnly` does not provide the ongoing automatic remediation required by the stated goal; therefore, the proposed solution does not meet the requirement.

Microsoft documents the Local Configuration Manager modes, including the automatic correction behavior of `ApplyAndAutoCorrect`, in Configuring the Local Configuration Manager. The registration cmdlet accepts the configuration mode as part of DSC node registration, as described in the Register-AzureRmAutomationDscNode reference.

QUESTION NO: 12

You are designing a strategy to monitor the baseline metrics of Azure virtual machines that run Windows Server. You need to collect detailed data about the processes running in the guest operating system. Which two agents should you deploy? Each correct answer presents part of the solution. NOTE: Each correct selection is worth one point.

A. the Dependency agent

B. the Azure Network Watcher Agent for Windows

C. the Telegraf agent

D. the Azure Log Analytics agent

ANSWER: A D

Explanation:

The Dependency agent and the Azure Log Analytics agent are the required pair in the classic Azure Monitor VM insights architecture represented by this question. The Azure Log Analytics agent, formerly called the Microsoft Monitoring Agent, collects guest operating-system telemetry from Windows Server virtual machines and sends it to a Log Analytics workspace. This includes data such as performance counters and Windows events, providing the guest-level collection channel that Azure Monitor uses for analysis.

The Dependency agent extends that collection by discovering processes running on the virtual machine and observing their network connections. It supplies the process and connection telemetry used to create VM insights dependency maps, allowing teams to understand which processes communicate with other machines, services, and ports. Together, these agents provide the detailed in-guest process visibility requested, rather than only Azure platform metrics.

For current deployments, Microsoft recommends the Azure Monitor Agent rather than the retired Log Analytics agent; however, the Azure Log Analytics agent is the correct selection from the provided choices and for the classic VM insights agent model. Microsoft documents the Dependency agent's role in collecting process and dependency information and its integration with Azure Monitor: Dependency agent for VM insights and Overview of VM insights.

QUESTION NO: 13 - (SIMULATION)

You need to configure access to Azure DevOps Agent pools to meet the forwarding requirements:

• Use a project agent pool when authoring build release pipelines.

• View the agent pool and agents of the organization.

• Use the principle of least privilege.

Which role memberships are required for the Azure 0e%Oos organization and the project? To answer, drag the appropriate role membership to the correct targets. Each role membership may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to content

NOTE: Each correct selection is worth one point.



ANSWER: See the explanation for the answer

Explanation:

Assign the following agent-pool roles:

Organization: `Reader`
Project: `User`

The `Reader` role at the organization level permits viewing the organization's agent pools and agents. The `User` role for the pool in the project permits using that pool when authoring build and release pipelines, without granting administrative permissions.

QUESTION NO: 14

You use Azure Repos for source control.

You need to configure the main branch to meet the following requirements:

• Changes must be reviewed by at least two team members before they are merged.

• Each pull request must be associated with an Azure Boards work item.

Which two branch policies should you configure? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Minimum number of reviewers

B. Check for linked work items

C. Require a merge strategy

D. Limit merge types

E. Automatically include code reviewers

ANSWER: A B

Explanation:

Configure the **Minimum number of reviewers** policy and set the required reviewer count to two. This policy prevents completion of a pull request until the required number of reviewers has approved the changes.

Configure the **Check for linked work items** policy. This policy requires the pull request to include at least one linked work item before the pull request can be completed. It provides traceability between the code change and the planned, tracked, or defect-related work in Azure Boards. See Improve code quality with branch policies.

QUESTION NO: 15

You use Git for source control.

You need to commit a 3-G3 ZIP file that contains virtual machines used for testing. The solution must meet the following requirements:

Which two actions should you include in the solution? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Store files in Azure Storage and enable blob versions.

B. Install the Git LFS extension and associate the extension to ZIP files.

C. Use GZip to compress the file before committing the file.

D. Install the git-stash extension and associate the extension to ZIP files.

E. Install the git-fat extension and associate the extension to ZIP files.

ANSWER: B E

Explanation:

Installing the Git LFS extension and associating ZIP files with it is appropriate because Git Large File Storage replaces the large ZIP content in the Git repository with a small pointer file. The actual binary payload is stored in LFS storage, which prevents a multi-gigabyte virtual-machine archive from becoming a normal Git object and unnecessarily inflating repository clones and history operations. Configure tracking for the ZIP pattern, commit the resulting `.gitattributes` configuration, and then commit the archive as usual. Azure Repos supports Git LFS and Microsoft recommends it for managing large binary assets in Git repositories. See Manage and store large files in Git.

Installing the git-fat extension and associating ZIP files with it also meets the large-file-management need. git-fat uses Git filters and attributes so that the repository records managed file content without treating the full large binary as an ordinary Git object in the repository database. This external-content approach is designed for large, non-diffable artifacts such as VM ZIP archives and retains a file-oriented workflow for contributors. The project's usage and attribute-based configuration are documented at git-fat on GitHub.

QUESTION NO: 16

You manage projects by using Azure Boards. You manage project code by using GitHub. You have a work itemthat has an ID of 456. You need to link work item 456 to a new pull request.

What are two ways to achieve this goal? Each correct answer presents a complete solution. NOTE: Each correct solution is worth one point.

A. To the description of the pull request, add #AB456.

B. To work item 456, add a comment that includes the URL of the pull request.

C. In the Development section for work item 456. select Add link, and then enter the URL of the pull request

D. From work item 456, open the Links tab, select Addlink, select Existing item and then enter the URL of the commit.

E. To the description of the pull request, add AB#456.

ANSWER: C E

Explanation:

To create a traceable Azure Boards–GitHub relationship, the pull request can include the Azure Boards work-item mention syntax **AB#456** in its description. When Azure Boards is connected to the relevant GitHub repository, it recognizes this syntax and associates the pull request with work item 456. The integration supports this convention for GitHub commits, issues, and pull requests, giving team members visibility of development activity directly from the work item.

The relationship can also be created from Azure Boards itself. In the **Development** section of work item 456, selecting **Add link** and providing the GitHub pull request URL creates the development link to that pull request. This is useful when the pull request already exists or when the person updating the work item needs to establish the relationship without editing the pull request description. Both approaches result in a GitHub pull-request development association that Azure Boards can display and track on the work item. Microsoft documents the supported **AB#** syntax and the workflow for linking GitHub development artifacts to Azure Boards work items in Link GitHub commits, pull requests, and issues to work items and Connect Azure Boards to GitHub.

QUESTION NO: 17

Your company has 60 developers who are assigned to four teams. Each team has 15 members.

The company uses an agile development methodology.

You need to structure the work of the development teams so that each team owns their respective work while working together to reach a common goal.

Which parts of the taxonomy should you enable the team to perform autonomously?

A. Features and Tasks

B. Initiatives and Epics

C. Epics and Features

D. Stories and Tasks

ANSWER: D

Explanation:

Stories and Tasks is correct because these backlog levels represent the work that an individual agile team should plan, decompose, deliver, and track independently. User stories express discrete customer requirements or valuable increments of functionality that a team can take into a sprint. Tasks then let team members break a selected story into the specific implementation, testing, documentation, and other activities necessary to complete it. Giving each of the four teams ownership at these levels supports autonomous backlog refinement, sprint planning, daily coordination, and delivery without requiring another team to manage its day-to-day work.

Azure Boards supports this model through team-specific backlogs, boards, capacity planning, and sprint views. A team can filter and manage the stories and tasks assigned to its area while still contributing to shared outcomes. The organization can coordinate the common goal through higher-level portfolio planning and hierarchy, while delivery teams retain responsibility for the work they directly execute. This separation provides alignment without removing team autonomy, which is a core objective when scaling agile work across multiple teams. Microsoft describes the relationship between backlog levels and work-item types in its Backlogs, boards, and plans documentation and explains how teams use work items and work item types to track development work.

QUESTION NO: 18

You have a YAML pipeline in Azure Pipelines.

The pipeline runs a test task and then publishes diagnostic log files.

You need to ensure that the diagnostic log files are published even if the test task fails or the pipeline is canceled.

Which condition should you use for the publishing task?

A. always()

B. succeeded()

C. failed()

D. succeededOrFailed()

ANSWER: A

Explanation:

Use **always()**. The `always()` condition evaluates to true even when an earlier task fails or the pipeline is canceled. This makes it appropriate for cleanup, diagnostics, test-result publication, and log-publication tasks that must run regardless of the result of earlier tasks.

The `succeeded()` condition would skip the task after a failure. The `failed()` condition runs only after a failure and would not publish the logs for successful runs. The `succeededOrFailed()` condition does not run when the pipeline is canceled. See Specify conditions.

QUESTION NO: 19

Your company implements an Agile development methodology.

You plan to implement retrospectives at the end of each sprint.

Which three questions should you include? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Who performed well?

B. Who should have performed better?

C. What could have gone better?

D. What went well?

E. What should we try next?

ANSWER: C D E

Explanation:

“What could have gone better?”, “What went well?”, and “What should we try next?” are appropriate Sprint Retrospective questions because they guide the team through a constructive continuous-improvement cycle. The team first recognizes successful practices and outcomes worth retaining, then examines process, collaboration, tools, and delivery issues that limited effectiveness during the sprint. Finally, it identifies a practical experiment or improvement action to apply in the next sprint. This structure encourages collective ownership and focuses the conversation on improving the way the team works rather than evaluating individual people. The Scrum Guide defines the Sprint Retrospective as an opportunity for the Scrum Team to inspect how the last Sprint went with respect to individuals, interactions, processes, tools, and its Definition of Done, and to plan improvements that increase quality and effectiveness. Azure DevOps supports this iterative Agile approach by enabling teams to track work, inspect delivery outcomes, and continuously adapt their processes. A useful retrospective should result in one or more clear, actionable changes that the team can carry into its next iteration. See the Scrum Guide and Microsoft’s Agile process workflow guidance.

QUESTION NO: 20

You need to configure GitHub to use Azure Active Directory (Azure AD) for authentication. What should you do first?

A. Create a conditional access policy in Azure AD.

B. Modify the Security settings of the GitHub organization.

C. Create an Azure Active Directory B2C (Azure AD B2C) tenant.

D. Register GitHub in Azure AD.

ANSWER: D

Explanation:

Register GitHub in Azure AD is the correct first step. For GitHub Enterprise Cloud authentication through Microsoft Entra ID (formerly Azure AD), the organization must first establish the GitHub Enterprise Cloud application in its Entra tenant. Typically, an administrator adds the preintegrated GitHub Enterprise Cloud application from the enterprise application gallery. This creates the application's service-principal configuration in the tenant and provides the place to configure SAML-based single sign-on.

Once the application exists, the administrator can configure the SAML values and obtain the identity provider information that GitHub requires, such as the sign-on URL, issuer, and signing certificate. Users and groups can also be assigned to the enterprise application where assignment is required. The GitHub enterprise's SAML settings are then populated with the Entra ID details to establish the trust relationship and direct authentication requests to the organization's identity provider. This sequence ensures that the identity-provider configuration and metadata are available before the GitHub-side SAML connection is completed.

Microsoft documents this process in its Microsoft Entra SSO integration with GitHub Enterprise Cloud tutorial, beginning with adding GitHub Enterprise Cloud from the gallery. See also Add an enterprise application in Microsoft Entra ID.

QUESTION NO: 21

You need to recommend a Docker container build strategy that meets the following requirements

• Minimizes image sues

• Minimizes the security surface area of the final image

What should you include m the recommendation?

A. multi-stage builds

B. single-stage builds

C. PowerShell Desired State Configuration (DSC)

D. Docker Swarm

ANSWER: A

Explanation:

multi-stage builds is the appropriate Docker build strategy because it separates the tools needed to compile, test, or publish an application from the minimal artifacts needed to run it. A Dockerfile can use an initial build stage containing SDKs, compilers, package managers, and source code, then copy only the compiled output into a later runtime stage. The resulting final image excludes development dependencies, intermediate files, build caches, credentials that might have been used during build operations, and other unnecessary content. This substantially reduces image size while also limiting the software and components present in the deployed container, thereby reducing its security surface area. Multi-stage builds are defined through multiple `FROM` instructions in one Dockerfile and commonly use `COPY --from=...` to transfer the required artifacts into the final stage. This approach also keeps the build process maintainable in a single Dockerfile while

producing a lean production image. See the Docker documentation on multi-stage builds and Microsoft guidance for building optimized .NET container images.

QUESTION NO: 22 - (DRAG DROP)

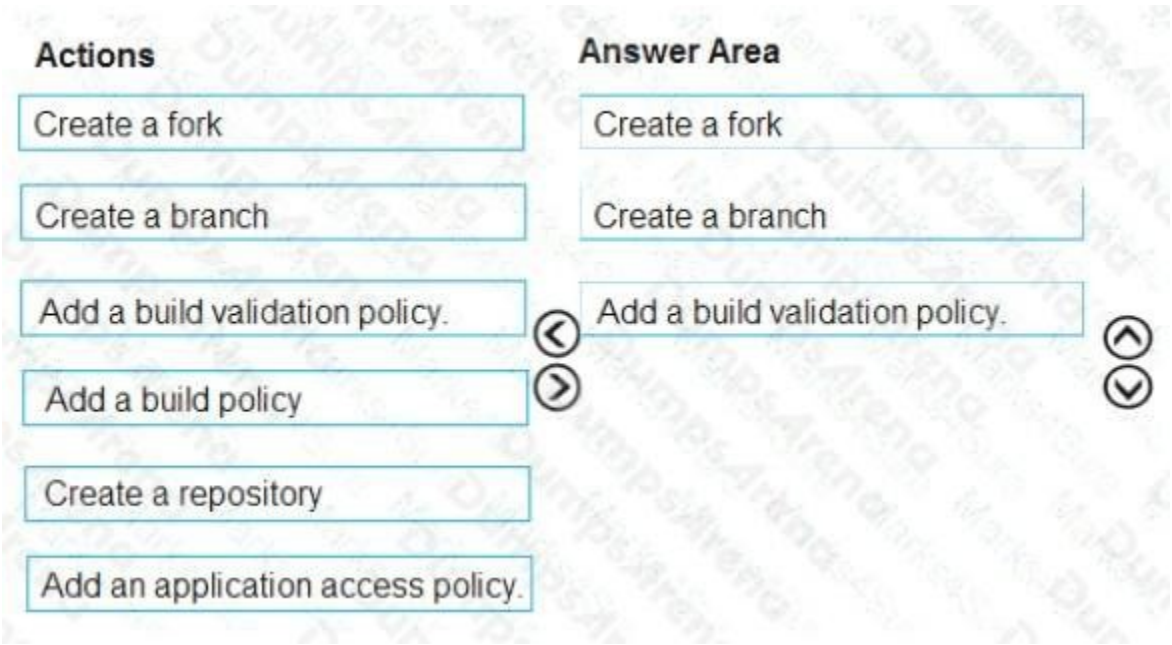
DRAG DROP

You need to implement the code flow strategy for Project2 in Azure DevOps.

Which three actions should you perform in sequence? To answer, move the appropriate actions from the list of actions to the answer area and arrange in the correct order.



ANSWER:



Explanation:

The required sequence is to first create a fork, then create a branch, and finally add a build validation policy. A fork provides Team2 with its own server-side copy of Project2 while preserving the connection to the original repository, known as the upstream repository. This is the appropriate Azure Repos pattern when a team must work independently but still submit its completed work back through pull requests. The fork is therefore created before development work begins.

Once the fork exists, Team2 creates a branch in that fork. The branch gives the team an isolated line of development for its changes, keeping work separate from the fork's primary branch until it is ready to be reviewed and integrated. Pull requests can then be used to propose the branch changes through the intended workflow.

After the target branch for pull requests has been established, a build validation policy is applied to it. Azure DevOps branch policies enforce pull-request quality requirements. A build validation policy queues an associated build when a pull request is created or updated and requires a successful result before the pull request can be completed. This ensures that the independent development workflow uses automated validation before code is merged. Microsoft's Azure Repos fork documentation describes forks and upstream pull-request workflows, while the branch policies documentation explains how build validation enforces successful builds for pull requests.

QUESTION NO: 23

You have an Azure DevOps organization named Contoso, an Azure DevOps project named Project1, an

Azure subscription named Sub1, and an Azure key vault named vault1.

You need to ensure that you can reference the values of the secrets stored in vault1 in all the pipelines of

Project1. The solution must prevent the values from being stored in the pipelines.

What should you do?

A. Create a variable group in Project1.

B. Add a secure file to Project1.

C. Modify the security settings of the pipelines.

D. Configure the security policy of Contoso.

ANSWER: A

Explanation:

Create a variable group in Project1 and configure it to link to Azure Key Vault vault1. Azure Pipelines variable groups are project-level library resources, so an authorized variable group can be referenced by multiple YAML or classic pipelines in the same project. When the variable group is linked to an Azure Key Vault, Azure DevOps obtains the selected secret values from the vault at pipeline runtime rather than requiring those values to be written into pipeline YAML, scripts, or pipeline definitions.

The variable group uses an Azure Resource Manager service connection that has permission to read secrets from the vault. After linking the vault and selecting the required secret names, pipelines can import the group and use the corresponding variables. The secret values remain protected: Azure Pipelines masks secret values in logs and the Key Vault-backed variable group stores secret names and the vault linkage, not the secret values themselves. Access to the variable group should be authorized only for the pipelines that need it, while Key Vault access is controlled through its configured authorization model. Microsoft documents the setup and behavior of Key Vault-linked variable groups in Use Azure Key Vault secrets in Azure Pipelines and describes variable groups as reusable, project-scoped pipeline variables in Manage variable groups.

QUESTION NO: 24

You have a build pipeline in Azure Pipelines that occasionally fails.

You discover that a test measuring the response time of an API endpoint causes the failures.

You need to prevent the build pipeline from failing due to The test.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point

A. Enable Test Impact Analysis (TIA).

B. Enable test slicing.

C. Clear Flaky tests included in test pass percentage

D. Set Flaky test detection to Off

E. Manually mark the test as flaky.

ANSWER: C E

Explanation:

An API response-time test can be sensitive to transient conditions such as shared infrastructure load, network latency, service warm-up, or temporary dependency delays. When the test is known to be unreliable as an indicator of an actual product regression, **Manually mark the test as flaky.** This classifies the specific test in Azure DevOps as flaky while retaining its results for investigation and future remediation. Flaky-test classification lets teams distinguish an intermittent test failure from a standard test failure in test reporting and analytics. In addition, **Clear Flaky tests included in test pass percentage** so that tests identified as flaky are excluded from the pass-percentage calculation. This prevents the known unstable response-time test from lowering the reported test pass rate and unnecessarily affecting the build's quality signal. Together, these actions preserve visibility into the test and its intermittent behavior without allowing that recognized flaky result to distort pipeline test-quality reporting. The team should still investigate and stabilize the API performance test over time, because flaky classification is a management mechanism rather than a replacement for reliable testing. See Manage flaky tests in Azure Pipelines and Review continuous test results after a build.

QUESTION NO: 25

You have a GitHub repository that contains workflows. The workflows contain steps that execute predefined actions. Each action has one or more versions.

You need to request the specific version of an action to execute.

Which three attributes can you use to identify the version? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. the SHA-based hashes

B. the tag

C. the runner

D. the branch

E. the serial

ANSWER: A B D

Explanation:

GitHub Actions selects an action version by placing a reference after the `@` in the action identifier, for example, `actions/checkout@v4`. The supported references include a commit SHA, a tag, and a branch. Therefore, `the SHA-based hashes`, `the tag`, and `the branch` can each identify the version to execute. A commit SHA identifies one immutable commit and is the most reproducible way to reference a third-party action. GitHub recommends pinning actions to a full-length commit SHA for stronger supply-chain security, because a SHA resolves to the exact action code that was reviewed. Tags are commonly used by action publishers to expose releases, including major-version tags such as `v4` and specific release tags. A branch is also a valid Git reference, such as `main` or a release branch, and GitHub resolves it to the commit currently at that branch reference when the workflow runs. GitHub's workflow syntax defines the action reference as a Git ref, SHA, or Docker tag, and its security guidance explains the benefit of full-length commit-SHA pinning. See GitHub Actions workflow syntax and GitHub security hardening guidance.

QUESTION NO: 26

Your company has an on-premises Bitbucket Server that is used for Git-based source control. The server is protected by a firewall that blocks inbound Internet traffic.

You plan to use Azure DevOps to manage the build and release processes.

Which two components are required to integrate Azure DevOps and Bitbucket? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. a deployment group

B. a Microsoft-hosted agent

C. service hooks

D. a self-hosted agent

E. an External Git service connection

ANSWER: D E

Explanation:

A self-hosted agent is required because the Bitbucket Server is reachable only from within the corporate network. Azure Pipelines agents initiate outbound communication to Azure DevOps to obtain jobs; therefore, an agent installed on a network-connected machine can receive a pipeline job and then clone or fetch source from the internal Bitbucket Server. This avoids any need to permit inbound Internet connections through the firewall. The agent identity and its execution environment must have network access to the Bitbucket endpoint and any required repository credentials.

An External Git service connection is required to register the on-premises, non-Azure Repos Git repository with Azure DevOps and provide the endpoint and authentication details that Azure Pipelines uses to access it. Azure DevOps supports external Git repositories through a service connection, enabling the pipeline configuration to identify the repository while the self-hosted agent performs the actual network access from the private network. Together, these components establish both the authenticated repository integration and the required private-network connectivity. Microsoft's guidance on external Git repository pipeline configuration is available at Pipeline options for Git repositories; agent connectivity and self-hosted agent behavior are documented at Azure Pipelines agents.

QUESTION NO: 27

You use GitHub for source control of .NET applications.

You need to deploy a documentation solution that meets the following requirements:

Which two tools can you use to compile the website? Each correct answer presents a complete solution.

A. JekyII

B. Medium

C. caret

D. WordPress

E. DocFX

F. Jekyll

ANSWER: E F

Explanation:

DocFX and **Jekyll** can each compile documentation source into a static website suitable for publishing from a GitHub-based workflow. DocFX is especially well suited to .NET documentation because it builds a static site from Markdown conceptual content and can generate API reference documentation from .NET source code, assemblies, or metadata. A DocFX project is configured with files such as `docfx.json`, then built during a local or CI workflow to produce deployable HTML assets. This lets a team maintain both product guides and API documentation alongside application source in GitHub. The DocFX documentation describes DocFX as a documentation generator for .NET that creates static documentation sites.

Jekyll is a static-site generator that transforms Markdown content, templates, layouts, and configuration into a static website. It is a natural fit for repository-managed documentation and is directly supported by GitHub Pages, which can build and publish Jekyll sites from a repository. This provides a straightforward route for compiling and hosting documentation when API-reference extraction is not required. GitHub documents the Jekyll integration and its use for GitHub Pages at About GitHub Pages and Jekyll.

QUESTION NO: 28

Your company uses Azure DevOps to manage the build and release processes for applications.

You use a Git repository for applications source control.

You plan to create a new branch from an existing pull request. Later, you plan to merge the new branch and the target branch of the pull request.

You need to use a pull request action to create the new branch. The solution must ensure that the branch uses only a portion of the code in the pull request.

Which pull request action should you use?

A. Set as default branch

B. Approve with suggestions

C. Cherry-pick

D. Reactivate

E. Revert

ANSWER: C

Explanation:

Cherry-pick is the appropriate pull request action because it copies selected commits from an existing pull request onto a branch chosen as the destination. In Azure Repos, using the Cherry-pick action from a pull request opens a workflow in which the user selects the target branch and can create a new branch for the copied changes. This enables the team to take only the commits that represent the required portion of the pull request, rather than completing the full pull request into its original target branch.

The resulting branch contains the selected changes applied to the specified base branch. The team can review, validate, and then open a separate pull request to merge that branch into the intended target branch. This is useful when an independent fix or limited feature segment must be promoted without including all work currently contained in the original pull request. Cherry-picking also preserves normal Git review and merge practices because the newly created branch can be evaluated through its own pull request before integration.

Microsoft documents cherry-picking as a way to copy changes from one branch to another and describes the Azure Repos pull-request experience for cherry-picking changes to a selected branch. See Copy changes with cherry-pick and Pull requests in Azure Repos.

QUESTION NO: 29

You deploy an Azure web app by using deployment slots.

The production slot and a staging slot use different connection strings.

You need to deploy a new version to the staging slot, validate it, and then move the application version to production without causing the connection strings to move between slots.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Configure each connection string as a deployment slot setting.

B. Swap the staging slot with the production slot.

C. Enable Azure App Service backup for both slots.

D. Configure the connection strings as general application settings.

E. Restart the production slot before the swap.

ANSWER: A B

Explanation:

Configure each connection string as a **deployment slot setting**. Slot settings are sticky, meaning that they remain with their current slot during a swap. This prevents a production connection string from being moved to staging or a staging connection string from being moved to production.

Swap the staging slot with the production slot after validation completes. A slot swap exchanges the application content and applicable non-sticky settings between the slots, allowing the validated application version to become live while the slot-specific connection strings remain associated with their respective environments. See Swap deployment slots and Configure deployment slot settings.

QUESTION NO: 30

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You need to recommend an integration strategy for the build process of a Java application. The solution must meet the following requirements:

â€¢ The builds must access an on-premises dependency management system.

â€¢ The build outputs must be stored as Server artifacts in Azure DevOps.

â€¢ The source code must be stored in a Git repository in Azure DevOps.

Solution: Configure the build pipeline to use a Hosted Ubuntu agent pool. Include the Java Tool Installer task in the build pipeline. Does this meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct because the essential requirement is for the build agent to access a dependency-management system hosted on the organization's on-premises network. Microsoft-hosted Ubuntu agents run in Microsoft-managed Azure infrastructure and are not placed within the organization's private network. Therefore, they do not inherently have network connectivity to private, on-premises endpoints that are available only on the internal network. Installing a Java version with the Java Tool

Installer task prepares the agent to run a Java build, but it does not establish the required private network path to the dependency system.

The appropriate integration approach is to use a self-hosted agent installed on a machine that can reach the on-premises dependency-management service, such as a machine within the corporate network or one connected through the organization's approved private networking configuration. A self-hosted agent can still connect outbound to Azure DevOps, check out source from Azure Repos Git, execute the Java build, and publish its output as Azure DevOps server artifacts. Microsoft describes self-hosted agents as suitable when builds require access to resources that are not available to Microsoft-hosted agents. See Microsoft documentation on self-hosted agents and Microsoft-hosted agent documentation.

QUESTION NO: 31

You manage code by using GitHub.

You need to ensure that repository owners are notified if a new vulnerable dependency or malware is found in their repository.

What should you do?

A. Configure branch protection rules for each repository.

B. Configure Dependabot alerts.

C. Configure CodeQL scanning actions.

D. Subscribe all the repository owners to the GitHub Advisory Database .

ANSWER: B

Explanation:

Configure Dependabot alerts. Dependabot alerts continuously compare the dependency manifests and lock files in a repository with security advisories in GitHub's Advisory Database. When GitHub identifies that a dependency used by the repository is affected by a newly published vulnerability or malware advisory, it creates an alert in the repository's Security view. Repository administrators and organization security managers can then review the affected package, the vulnerable version range, and the dependency path to prioritize remediation.

Dependabot alert notifications can be delivered to the relevant repository and organization stakeholders through GitHub notification mechanisms, helping ensure that newly disclosed supply-chain risks receive attention without requiring manual monitoring of advisory publications. GitHub also supports alerting for malicious packages, including advisories for packages identified as malware, when the repository depends on an affected package. This makes Dependabot alerts the GitHub capability designed specifically for detecting and notifying maintainers about vulnerable or malicious dependencies already present in a repository. For configuration details and notification behavior, see About Dependabot alerts and Configuring Dependabot alerts.

QUESTION NO: 32

You have the following Azure policy.



A. ensures that at) data for new Azure Storage accounts is encrypted at rest

B. prevents HTTPS traffic to new Azure Storage accounts when the accounts are accessed over the internet

C. prevents all HTTP traffic to wasting Azure Storage accounts

D. ensures that all traffic to new Azure Storage accounts is encrypted

ANSWER: D

Explanation:

The policy ensures that all traffic to new Azure Storage accounts is encrypted because it evaluates the Storage account secure-transfer setting, exposed in Azure Resource Manager as `supportsHttpsTrafficOnly`. A policy rule that targets `Microsoft.Storage/storageAccounts` and uses a `Deny` effect when this property is not enabled prevents a noncompliant Storage account from being created or updated. Consequently, newly provisioned accounts governed by this policy must require secure transfer.

Secure transfer required protects data in transit by requiring clients to use secure connections when accessing Azure Storage. For REST-based access, this means HTTPS rather than HTTP. Azure Storage also applies secure transport requirements appropriate to supported file-sharing protocols. This configuration is an account-level control, so it establishes the secure-transfer requirement for requests to the governed Storage account rather than relying on each individual client to select a secure endpoint. The Azure Policy deny effect enforces this at resource deployment time, making the requirement effective before an insecure account configuration can be deployed. Microsoft documents secure transfer requirements at Require secure transfer to ensure secure connections and describes enforcement behavior for the Azure Policy Deny effect.

QUESTION NO: 33

You use Azure Pipelines to manage project builds and deployments.

You plan to use Azure Pipelines for Microsoft Teams to notify the legal team when a new build is ready for release. You need to configure the Organization Settings in Azure DevOps to support Azure Pipelines for Microsoft Teams. What should you turn on?

A. Azure Active Directory Conditional Access Policy Validation

B. Alternate authentication credentials

C. Third-party application access via OAuth

D. SSH authentication

ANSWER: C

Explanation:

Third-party application access via OAuth must be enabled because the Azure Pipelines app for Microsoft Teams is an external application that connects to Azure DevOps by using OAuth authorization. OAuth lets a user grant the Teams app appropriately scoped access to Azure DevOps resources without exposing the user's password. This authorization is necessary for the app to access pipeline information and establish subscriptions that deliver build and release notifications to the relevant Teams channel.

In Azure DevOps, this is an organization-level application access policy. An organization administrator can manage it from *Organization settings*, under the policies governing application connections and OAuth access. If third-party application access through OAuth is disabled, the Azure Pipelines Teams integration cannot complete its authorization flow, even when the relevant users otherwise have permission to view the pipeline or project. After enabling the policy, users can authorize the Azure Pipelines app in Teams and configure notifications for build or release events.

Microsoft explicitly lists enabling this OAuth policy as a prerequisite for the Azure Pipelines integration with Microsoft Teams. See Azure Pipelines with Microsoft Teams and Change application connection and access policies.

QUESTION NO: 34

You store source code in a Git repository in Azure repos. You use a third-party continuous integration (CI) tool to control builds.

What will Azure DevOps use to authenticate with the tool?

A. certificate authentication

B. a personal access token (PAT)

C. a Shared Access Signature (SAS) token

D. NTLM authentication

ANSWER: B

Explanation:

A personal access token (PAT) is the appropriate credential for a third-party CI integration that needs authenticated access to an Azure Repos Git repository. A PAT is an Azure DevOps credential intended for tools, automation, and service accounts where interactive user sign-in is not suitable. The CI tool can use the token when it clones or fetches repository content and, where needed, when it calls Azure DevOps REST APIs.

The token should be created for a dedicated automation identity where possible and granted only the scopes required for the integration. For a CI system that only retrieves source code, the relevant least-privilege scope is typically *Code (Read)*. Azure DevOps also supports token expiration and revocation, allowing the organization to rotate or disable the credential without changing a user's password. Store the PAT in the CI tool's secure credential store rather than embedding it in pipeline definitions or repository files.

Microsoft documents PATs as an authentication method for Azure DevOps resources and provides specific guidance for Git authentication to Azure Repos. See Use personal access tokens and Authentication overview for Azure Repos Git.

QUESTION NO: 35

You use GitHub for source control.

A file that contains sensitive data is committed accidentally to the Git repository of a project.

You need to delete the file and its history form the repository.

Which two tools can you use? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. the git filter-branch command

B. BFG Repo-Cleaner

C. the git rebase command

D. GitHub Desktop

ANSWER: A B

Explanation:

the git filter-branch command and **BFG Repo-Cleaner** can both rewrite the repository's commit history to remove a sensitive file from every commit in which it exists. This is necessary because deleting the file through a normal commit removes it only from the latest version of the repository; the sensitive content remains reachable through prior commits. After rewriting history, the updated refs must be force-pushed so that the remote repository reflects the rewritten commit graph.

the git filter-branch command is a native Git history-rewriting utility and can filter all revisions to remove a specified path. Although modern Git guidance generally prefers newer tooling for routine use, filter-branch can perform the required full-history removal. **BFG Repo-Cleaner** is a purpose-built history-cleaning tool that can efficiently remove files and sensitive

content from Git history. GitHub documents BFG as a supported approach for removing sensitive data. Any exposed secret should also be revoked or rotated, because it may have been cloned, cached, or accessed before the history rewrite. See GitHub: Removing sensitive data from a repository and BFG Repo-Cleaner documentation.

QUESTION NO: 36

Your company creates a web application.

You need to recommend a solution that automatically sends to Microsoft Teams a daily summary of the exceptions that occur in the application.

Which two Azure services should you recommend? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Azure Logic Apps

B. Azure Pipelines

C. Microsoft Visual Studio App Center

D. Azure DevOps Project

E. Azure Application Insights

ANSWER: A E

Explanation:

Azure Application Insights collects and stores telemetry from the web application, including handled and unhandled exception data. Its exception telemetry and Log Analytics/Kusto Query Language querying capabilities enable the organization to identify exceptions over a defined daily time range and produce the data required for a summary. Application Insights is an Azure Monitor feature designed specifically to monitor the availability, performance, usage, failures, and dependencies of applications.

Azure Logic Apps supplies the workflow automation and Microsoft Teams integration needed for delivery. A logic app can use a Recurrence trigger to run once each day, execute a query against the Application Insights-backed logs, shape the returned exception information into a readable summary, and publish that summary to a Teams chat or channel through the Microsoft Teams connector. This creates a scheduled, no-code or low-code operational notification process rather than requiring someone to inspect telemetry manually. Microsoft documents exception collection and analysis in Application Insights exception diagnostics and documents how to use the Teams connector from Azure Logic Apps at Microsoft Teams connector for Azure Logic Apps.

QUESTION NO: 37

You have an Azure subscription. The subscription contains virtual machines that run either Windows Server or Linux.

You plan to use Prometheus to monitor performance metrics.

You need to integrate Prometheus and Azure Monitor.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Install a Prometheus server on a Windows virtual machine in Azure.

B. On each virtual machine, expose the metrics endpoint.

C. On each virtual machine, enable the Azure Diagnostics extension.

D. On each virtual machine, enable the containerized agent for Azure Monitor.

E. Expose a virtual network service endpoint for Azure Storage.

F. Install a Prometheus server on a Linux virtual machine in Azure.

ANSWER: B F

Explanation:

On each virtual machine, expose the metrics endpoint. is required because Prometheus uses a pull-based model. A Prometheus server retrieves measurements by scraping an HTTP endpoint exposed by each monitored target. For virtual-machine performance monitoring, the endpoint is normally supplied by an appropriate exporter or by an instrumented application. The Prometheus server configuration identifies each target and its metrics path, allowing the collected measurements to be queried, alerted on, and forwarded through the selected Azure Monitor integration workflow.

Install a Prometheus server on a Linux virtual machine in Azure. provides the Prometheus scrape server and the central configuration point for the virtual-machine targets. Prometheus is commonly deployed on Linux, where it can run continuously, discover or statically configure the Windows Server and Linux targets, collect their exposed metrics, and retain or relay the resulting time-series data. Azure Monitor provides a managed service for Prometheus that supports storing and analyzing Prometheus-compatible metrics, enabling Azure-based visualization and alerting scenarios after collection is configured. Microsoft describes the service and its Prometheus-compatible monitoring capabilities at Azure Monitor managed service for Prometheus. Prometheus also documents that exporters expose the metrics that Prometheus scrapes at Prometheus exporters and integrations.

QUESTION NO: 38

You are integrating an Azure Boards project and a GitHub repository.

You need to authenticate Azure Boards to GitHub.

Which two authentication methods can you use? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. a trusted root certificate

B. a publisher certificate

C. Azure Active Directory (Azure AD)

D. GitHub user credentials

E. a personal access token (PAT)

ANSWER: D E

Explanation:

GitHub user credentials and a personal access token (PAT) are supported ways to authorize the Azure Boards integration with GitHub. When an administrator connects Azure Boards to GitHub.com, they can authenticate by signing in to a GitHub account and completing GitHub's authorization flow. The GitHub identity used must have sufficient access to the organization and repositories being connected. This authorization enables Azure Boards to associate work items with GitHub commits, pull requests, and branches, and allows the integration to retrieve the repository information needed for those links.

A GitHub personal access token can also supply the required GitHub authorization. This is particularly applicable to GitHub Enterprise Server connections, where Azure DevOps requires a token created by a GitHub account that has the necessary repository and organization permissions. The token is used as the credential by which Azure Boards accesses the selected GitHub resources. In either case, access is granted through a GitHub user identity or through a token representing that GitHub identity; Azure AD and certificate types are not the authentication mechanisms used for this Azure Boards-to-GitHub connection. Microsoft documents the GitHub sign-in authorization process and the PAT requirements for GitHub Enterprise Server connections in Connect Azure Boards to GitHub and Connect Azure Boards to GitHub Enterprise Server.

QUESTION NO: 39

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen

Your company has a project in Azure DevOps for a new web application.

You need to ensure that when code is checked in, a build runs automatically.

Solution: From the Continuous deployment trigger settings of the release pipeline, you enable thePull request triggersetting.

Does the meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. The requirement is to run a build automatically when source code is checked in. In Azure DevOps, this is implemented by configuring a continuous integration trigger on the *build pipeline*. A CI trigger monitors the selected repository and branches, queues a new pipeline run when qualifying commits are pushed, and thereby validates the newly checked-in code through the defined build process.

Release-pipeline continuous deployment settings have a different purpose: they initiate deployment releases in response to successful build artifacts becoming available. Likewise, pull-request triggers are intended to validate proposed changes associated with pull requests; they do not configure a normal check-in-triggered build. Therefore, enabling a pull-request trigger under continuous deployment settings does not establish the required automatic build for code check-ins.

Configure the build pipeline's trigger settings instead. For YAML pipelines, define a `trigger` section with the applicable branches; for classic build pipelines, enable Continuous integration on the Triggers tab. Microsoft documents CI triggers and their branch filtering behavior at CI trigger schema and explains pipeline trigger configuration in Configure pipeline triggers.

QUESTION NO: 40

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You use Azure Pipelines to build and test a React.js application.

You have a pipeline that has a single job.

You discover that installing JavaScript packages from npm takes approximately five minutes each time you run the pipeline.

You need to recommend a solution to reduce the pipeline execution time.

Solution: You recommend defining a container job that uses a custom container that has the JavaScript packages preinstalled.

Does this meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. A custom container with JavaScript packages preinstalled does not reliably eliminate the dependency installation required by a React.js project. Project dependencies are normally restored according to the repository's `package.json` and lock file, into the job's workspace. This ensures that the exact dependency graph used for the build is reproducible and changes whenever the project's declared dependencies change. Preinstalling packages in a container image does not inherently populate the workspace-specific dependencies that npm expects for that particular source revision.

Azure Pipelines pipeline caching is the intended mechanism for reducing repeated dependency-download time between runs. A `Cache@2` task can cache npm's package cache, using a key that includes the lock file. When the lock file is unchanged, npm can reuse previously downloaded package artifacts; when dependencies change, the key changes and an appropriate new cache is created. This directly addresses the repeated five-minute npm restore cost while preserving dependency version control through the lock file. Microsoft documents pipeline caching specifically for reusing downloaded dependencies and provides npm-oriented JavaScript pipeline guidance. See Pipeline caching in Azure Pipelines and Build JavaScript apps with Azure Pipelines.

QUESTION NO: 41

TION NO: 9

You have a GitHub repository that contains workflows. The workflows contain steps that execute predefined actions. Each action has one or more versions.

You need to request the specific version of an action to execute.

Which three attributes can you use to identify the version? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. the SHA-based hashes

B. the tag

C. the runner

D. the branch

E. the serial

ANSWER: A B D

Explanation:

GitHub Actions identifies the revision of an action in the `uses` statement by appending a Git reference after `@`. A full commit SHA can identify an exact immutable revision, such as `actions/checkout@<commit-SHA>`. Pinning to a full-length commit SHA is GitHub's recommended approach for the strongest stability and security because the referenced code cannot be changed after the workflow is authored. A tag can also identify an action version, for example `actions/checkout@v4` or a more specific release tag. Tags are commonly used by action maintainers to represent published versions. Finally, a branch name is a valid Git reference and can be used to select the action code currently present on that branch, such as `owner/action@main`. Thus, SHA-based hashes, tags, and branches are all supported ways to request an action version or revision. GitHub recommends selecting a specific version rather than omitting the reference, because an unpinned action reference can unexpectedly change when its publisher updates the action. See the GitHub Actions workflow syntax documentation at GitHub Docs: jobs.<job_id>.steps[*].uses and GitHub's security guidance at Security hardening for GitHub Actions.

QUESTION NO: 42

You are currently defining a release strategy for an app, named APP-01.

The strategy should allow you to keep the time it takes to deploy new releases of the app to a minimum. The strategy should also allow you to roll back in the shortest time required.

Which of the following is the release strategy you should use?

A. Red/Black deployment

B. Rolling deployment

C. “Big Bang” deployment

D. Canary deployment

ANSWER: A

Explanation:

Red/Black deployment is the appropriate strategy because it maintains two production-capable environments: one runs the current live version, while the other is prepared with the new version. The new release can be deployed, configured, and validated in the inactive environment before any production traffic is moved. Release therefore consists primarily of switching traffic to the already-running new environment, making the production cutover very fast.

This approach also provides an exceptionally short rollback path. If an issue is discovered after traffic is switched, the previous environment remains available with the prior version still deployed. Traffic can simply be switched back, avoiding the time and risk involved in rebuilding, redeploying, or reversing application changes. In Microsoft Azure, Azure App Service deployment slots support this model: an application can be staged in a slot and swapped into production, while swap-back enables rapid recovery when necessary. Microsoft describes deployment slots and swap operations at Set up staging environments in Azure App Service. Microsoft’s Azure Architecture Center also describes blue-green deployment—the equivalent pattern commonly called red/black deployment—as using two identical production environments and switching routing between them: Blue-green deployment.

QUESTION NO: 43

You have a free tier of an Azure DevOps organization named Contoso. Contoso contains 10 private projects. Each project has multiple jobs with no dependencies. The build process requires access to resource files located in an on-premises file system.

You frequently run the jobs on five self-hosted agents but experience long build times and frequently queued builds.

You need to minimize the number of queued builds and the time it takes to run the builds.

What should you do?

A. Configure the pipelines to use the Microsoft-hosted agents.

B. Register additional self-hosted agents.

C. Purchase self-hosted parallel jobs.

D. Purchase Microsoft-hosted parallel jobs.

ANSWER: C

Explanation:

Purchase self-hosted parallel jobs. In Azure Pipelines, an agent is the machine or process that executes a job, whereas a parallel job is the licensed capacity that permits a pipeline job to run concurrently. Having five registered self-hosted agents does not by itself mean that five jobs can be scheduled at once: the organization must also have sufficient self-hosted parallel-job capacity. The frequent queueing indicates that available concurrency is the limiting factor.

Because the projects contain independent jobs, additional parallel-job capacity enables Azure Pipelines to dispatch more jobs simultaneously to the existing self-hosted agent pool. This reduces the time jobs wait in the queue and can reduce the overall elapsed build time through parallel execution. Self-hosted agents also remain appropriate for this workload because

they can run within, or have network connectivity to, the on-premises environment that contains the required resource files. Microsoft distinguishes agent availability from parallel-job entitlement and documents that self-hosted parallel jobs control the number of concurrent self-hosted pipeline jobs. See Configure and pay for parallel jobs and Azure Pipelines agents.

QUESTION NO: 44

Your development team is building a new web solution by using the Microsoft Visual Studio integrated development environment (IDE).

You need to make a custom package available to all the developers. The package must be managed centrally, and the latest version must be available for consumption in Visual Studio automatically.

Which three actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Publish the package to a feed.

B. Create a new feed in Azure Artifacts.

C. Upload a package to a Git repository.

D. Add the package URL to the Environment settings in Visual Studio.

E. Add the package URL to the NuGet Package Manager settings in Visual Studio.

F. Create a Git repository in Azure Repos.

ANSWER: A B E

Explanation:

Creating a new feed in Azure Artifacts establishes the centrally managed package repository required for the team. An Azure Artifacts feed is a NuGet-compatible source that stores packages and their versions, applies access controls, and provides a consistent location from which authorized developers can obtain internal dependencies. Publishing the package to a feed then places the custom package and its version metadata into that shared source, making it available to all developers rather than distributing package files manually.

Adding the package URL to the NuGet Package Manager settings in Visual Studio configures the Azure Artifacts feed as a NuGet package source. Visual Studio can then query the feed when developers browse, install, restore, or check for updates to packages. Newly published versions become available through the configured source without each developer having to locate or configure a separate package location. In practice, projects should use appropriate package-version management, such as centrally managed package versions or explicit version updates, to control when a newer available package version is adopted.

Microsoft documents the workflow of creating an Azure Artifacts feed, publishing NuGet packages to it, and connecting Visual Studio or NuGet clients to the feed for package consumption. See Get started with NuGet packages in Azure Artifacts and Consume NuGet packages from Azure Artifacts.

QUESTION NO: 45

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You use Azure Pipelines to build and test a React.js application.

You have a pipeline that has a single job.

You discover that installing JavaScript packages from npm takes approximately five minutes each time you run the pipeline.

You need to recommend a solution to reduce the pipeline execution time.

Solution: You recommend enabling pipeline caching.

Does this meet the goal?

A. Yes

B. No

ANSWER: A

Explanation:

Yes is correct. Azure Pipelines caching is intended to reduce execution time when a pipeline repeatedly downloads or restores dependencies that are expensive to obtain. In a React.js build, npm installation commonly takes several minutes on a fresh Microsoft-hosted agent because the agent must retrieve package tarballs again. A pipeline cache can persist the npm cache directory across runs and restore it early in a subsequent run. The cache key should include the operating system and the dependency lock file, such as `package-lock.json`, so a changed dependency graph produces a distinct cache. With the cache restored, `npm ci` or `npm install` can reuse the locally cached package content rather than downloading it from the registry, reducing the dependency-install portion of the job. Pipeline caching is especially suitable here because the application has a single job and the objective is specifically to speed up repeated npm package retrieval between pipeline executions. Microsoft's Cache@2 task documentation includes npm as a supported caching scenario and recommends using a lock file as part of the key to ensure cached contents correspond to declared dependencies. See Microsoft Learn: Pipeline caching and Microsoft Learn: JavaScript in Azure Pipelines.

QUESTION NO: 46

You have an Azure Resource Manager template that deploys a multi-tier application.

You need to prevent the user who performs the deployment from viewing the account credentials and connection strings used by the application.

What should you use?

A. Azure Key Vault

B. a Web.config file

C. an Appsettings.json file

D. an Azure Storage table

E. an Azure Resource Manager parameter file

ANSWER: A

Explanation:

Azure Key Vault is the appropriate service for protecting account credentials, connection strings, passwords, and other sensitive deployment values. Rather than placing these values directly in an Azure Resource Manager template or exposing them in a parameter file, store each value as a secret in a key vault. The deployment can then use a Key Vault reference to obtain the secret at deployment time. This keeps the secret value out of the template definition and avoids disclosing it in deployment inputs, deployment history, or logs viewed by the person initiating the deployment.

For this pattern, the deployment identity must have permission to retrieve the required secret from the vault. The template parameter should also use a secure type, such as `secureString`, so Azure Resource Manager treats the value as sensitive throughout deployment processing. Key Vault also provides centralized secret lifecycle management, access control through Azure RBAC, auditing, and the ability to rotate secrets without storing credentials in source control. Microsoft documents how to use a Key Vault secret as a secure parameter during an ARM deployment in Use Azure Key Vault to pass

secure parameter values during deployment. For Key Vault secret-management capabilities, see the Azure Key Vault overview.

QUESTION NO: 47

You have a GitHub repository that contains a workflow used to deploy resources to an Azure subscription.

You need to configure the workflow to authenticate to Azure by using OpenID Connect (OIDC). The solution must avoid storing an Azure client secret in GitHub.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Create a federated identity credential for the Azure application identity.

B. Grant the workflow the id-token: write permission.

C. Store the application client secret as an organization secret.

D. Grant the workflow the contents: write permission.

E. Create a personal access token (PAT) for the Azure subscription.

ANSWER: A B

Explanation:

Create a **federated identity credential** for the Microsoft Entra application or user-assigned managed identity used by the workflow. The federated credential establishes a trust relationship between Azure and the GitHub Actions workload identity. It can be scoped to a GitHub organization, repository, branch, environment, or other supported subject claim.

You must also grant the workflow the **id-token: write** permission. This permission allows GitHub Actions to request an OIDC token during the workflow run. The Azure Login action can exchange that token for an Azure access token without using a client secret. See Use OpenID Connect with GitHub Actions and About security hardening with OpenID Connect.

QUESTION NO: 48

You use Azure Pipelines to build and test code.

You need to analyze the agent pool usage.

What are two ways to achieve the goal? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. Review the historical graph for the agent pools.

B. Review the Pipeline duration report.

C. Query the TaskAgentPoolSizeSnapshot/TaskAgentPoolSizeSnapshots endpoint.

D. Query the PipelineRun/PipelineRuns endpoint.

ANSWER: A C

Explanation:

Review the historical graph for the agent pools is correct because Azure DevOps includes a built-in Pool consumption report for examining how an agent pool has been used over time. The report presents historical job and agent-capacity information, helping teams recognize periods of queueing, identify insufficient capacity, and make decisions about scaling self-hosted

agents or purchasing additional parallel jobs. It provides a direct portal-based solution for operational analysis without requiring a custom report.

Query the TaskAgentPoolSizeSnapshot/TaskAgentPoolSizeSnapshots endpoint is also correct because Azure DevOps Analytics exposes agent-pool snapshot data through its OData model. Snapshot records can be queried across time and incorporated into a custom dashboard, reporting process, or Power BI model. This enables programmatic analysis of trends in pool size and utilization, including historical capacity characteristics that are useful for planning agent availability. Microsoft documents the built-in reporting experience in the Pool consumption report documentation and lists the Analytics entities available for pipeline reporting in the Azure Pipelines Analytics entity reference.

QUESTION NO: 49

You have several Azure Active Directory (Azure AD) accounts.

You need to ensure that users use multi-factor authentication (MFA) to access Azure apps from untrusted networks.

What should you configure in Azure AD?

A. access reviews

B. managed identities

C. entitlement management

D. conditional access

ANSWER: D

Explanation:

Conditional access is the appropriate configuration because Microsoft Entra ID Conditional Access evaluates sign-in context and applies access requirements dynamically. A policy can target the required users or groups and the relevant Azure cloud applications, then use the Locations condition to identify sign-ins from networks that are not trusted. Administrators define trusted network ranges as named locations and configure the policy scope so that connections outside those trusted locations are subject to the policy. The policy's grant controls can then require multifactor authentication before access is granted.

This approach meets the requirement precisely: users are not simply enrolled for MFA globally; instead, the MFA requirement is enforced according to the network from which they attempt to sign in. Conditional Access is Microsoft's policy engine for making access decisions using signals including user or workload identity, application, device state, location, sign-in risk, and user risk. In current terminology, Azure AD is Microsoft Entra ID, but the Conditional Access capability and policy behavior remain the same. Implementing the policy should include appropriate exclusions for emergency access accounts and testing in report-only mode before enforcement. See the Microsoft Entra Conditional Access overview and Microsoft's guidance for using location conditions in Conditional Access policies.

QUESTION NO: 50

You have an Azure web app named webapp1 that uses the .NET Core runtime stack. You have an Azure Application Insights resource named AppInsight1. Webapp1 sends telemetry data to AppInsights1.

You need to ensure that webapp1 sends the telemetry data at a fixed sampling rate.

What should you do?

A. From the code repository of webapp1, modify the ApplicationInsights.config file.

B. From the code repository of webapp1, modify the Startup.cs file.

C. From AppInsights1. modify the Usage and estimated costs settings.

D. From AppInsights1, configure the Continuous export settings.

ANSWER: B

Explanation:

From the code repository of webapp1, modify the Startup.cs file. In an ASP.NET Core application, Application Insights telemetry is configured through dependency injection during service registration, traditionally in `Startup.cs` within `ConfigureServices`. To use a fixed sampling percentage, configure the Application Insights telemetry processor chain with the SDK's fixed-rate sampling processor. For example, the application can disable adaptive sampling and add a sampling processor with the required percentage through `TelemetryConfiguration`. This makes the sampling decision in the web app before telemetry is transmitted, providing a predictable, fixed proportion of requests, dependencies, traces, and other telemetry sent to the Application Insights resource. Fixed-rate sampling is especially useful when a known, consistent sampling percentage is required rather than a rate that automatically changes in response to application traffic volume. Microsoft documents configuring telemetry processors, including sampling, in ASP.NET Core during application service configuration. See Application Insights telemetry sampling and Application Insights for ASP.NET Core applications.

QUESTION NO: 51

You have an Azure subscription that contains multiple Azure services.

You need to send an SMS alert when scheduled maintenance is planned for the Azure services.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Enable Azure Security Center.

B. Create and configure an Azure Monitor alert rule.

C. Create an Azure Service Health alert.

D. Create and configure an action group.

ANSWER: C D

Explanation:

Create an Azure Service Health alert is required because Azure Service Health publishes notifications about Azure platform events that can affect a subscription, including planned maintenance. A Service Health alert can be configured to monitor the relevant subscription, Azure services, regions, and the Planned Maintenance event type. This gives the alerting system the event source and filtering needed to identify scheduled maintenance that applies to the organization's Azure resources.

Create and configure an action group is also required to deliver the notification by SMS. Azure Monitor action groups define the notification recipients and delivery channels used when an alert is triggered. An action group can contain an Email/SMS message/Push/Voice notification receiver, where an administrator supplies the recipient phone number and country code for SMS delivery. Associating that action group with the Service Health alert connects the planned-maintenance event to the required SMS notification. Microsoft documents Service Health alerts as the mechanism for alerts on service issues, planned maintenance, and health advisories, while action groups provide the notification actions used by Azure Monitor alerts. See Create Service Health alerts in the Azure portal and Azure Monitor action groups.

QUESTION NO: 52

You use Azure Pipelines to build and test code.

You need to analyze the agent pool usage.

What are two ways to achieve the goal? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. Review the historical graph for the agent pools.

B. Review the Pipeline duration report.

C. Query the TaskAgentPoolSizeSnapshot/TaskAgentPoolSizeSnapshots endpoint.

D. Query the PipelineRun/PipelineRuns endpoint.

ANSWER: A C

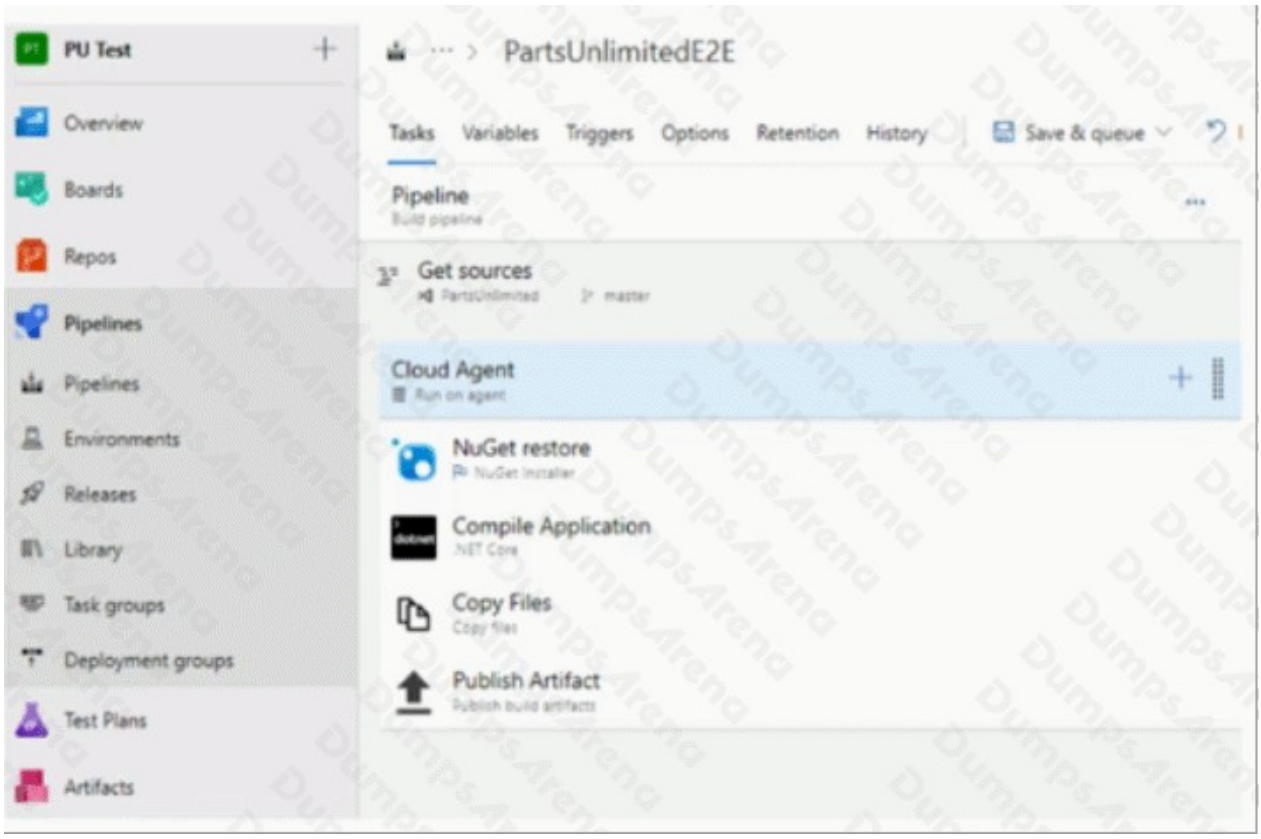
Explanation:

Review the historical graph for the agent pools. is correct because Azure DevOps provides an agent-pool usage view in the organization's Agent pools settings. This historical graph lets administrators inspect how jobs used the selected pool over time, including the relationship between available capacity and queued or running work. It is a direct, built-in way to identify capacity pressure and determine whether the pool needs additional agents.

Query the TaskAgentPoolSizeSnapshot/TaskAgentPoolSizeSnapshots endpoint. is also correct because Azure DevOps Analytics exposes task-agent pool size snapshot data through its OData entity set. Snapshot data can be queried and incorporated into custom reports or dashboards to analyze agent-pool capacity trends over time. This supports programmatic and historical analysis rather than requiring review solely in the Azure DevOps portal. The Analytics data model documents task-agent-related entities, while Azure DevOps documentation describes managing and monitoring agent pools. See Analytics entity reference for pipelines and agents and Azure Pipelines agent pools.

QUESTION NO: 53 - (HOTSPOT)

You have the Azure DevOps pipeline shown in the following exhibit.



Use the drop-down menus to select the answer choice that completes each statement based on the information presented in the graphic.



ANSWER:



Explanation:

The pipeline has 1 job and 4 tasks. A job is the unit of work that Azure Pipelines assigns to an agent. The exhibit shows a single agent job, labeled as the cloud agent job, which establishes one execution context for the listed build work. All of the steps shown beneath that agent-job container belong to the same job and run as part of its sequence.

Within that job, the exhibit displays four distinct tasks: restoring NuGet packages, compiling the application, copying files, and publishing the artifact. Each displayed operation is a separate task because it performs an individual build or delivery action within the job. Together, these tasks define the ordered work performed by the assigned build agent, while the job provides the common execution boundary for that work.

This reflects the Azure Pipelines model in which a pipeline can contain jobs, and each job contains one or more sequential steps. A task is a predefined packaged step that carries out a specific operation, such as building source code, copying output, or publishing build artifacts. Microsoft documents that jobs are collections of steps that run on an agent and that tasks are the building blocks used in a pipeline. See Azure Pipelines jobs and Azure Pipelines tasks. Thus, the highlighted selections accurately identify the one job and four contained tasks.

QUESTION NO: 54

You have an X.509 certificate named cert1 that is managed by an external partner named Partner 1. You build a web app named Web1 that will be deployed by using Azure Pipelines. Web1 will be bound to cert1. You need to ensure that cert1 is available to Azure Pipelines. The solution must ensure that cert1 can only be updated by Partner1. Where should you store cert1?

A. the Secure files library in Azure DevOps

B. a secure enclave

C. a variable group

D. an Azure key vault

ANSWER: D

Explanation:

an Azure key vault is the appropriate storage location because Azure Key Vault is designed to securely store and manage certificates, including X.509 certificates and their associated private keys. Azure Pipelines can retrieve a certificate from a

key vault during deployment by using an authorized Azure service connection or workload identity, avoiding the need to place sensitive certificate material directly in source control, pipeline YAML, or general-purpose variables.

Key Vault also supports the required separation of duties. Access can be controlled through Azure role-based access control or Key Vault access policies so that Partner1 receives only the certificate-management permissions needed to import, create, renew, or update cert1. The Azure Pipelines deployment identity can instead be assigned narrowly scoped read permissions to retrieve the certificate or secret when deploying Web1. This allows the pipeline to use the certificate without granting it authority to modify it, while preventing other users or identities from updating the certificate.

Key Vault additionally provides centralized security controls, auditing through Azure activity and diagnostic logs, and lifecycle support for certificates. These capabilities make it suitable for protecting deployment certificates managed by an external party. For more information, see Azure Key Vault certificates and Use Azure Key Vault secrets in Azure Pipelines.

QUESTION NO: 55

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You have an Azure DevOps organization named Contoso and an Azure subscription. The subscription contains an Azure virtual machine scale set named VMSS1 that is configured for autoscaling.

You have a project in Azure DevOps named Project1. Project1 is used to build a web app named App1 and deploy App1 to VMSS1.

You need to ensure that an email alert is generated whenever VMSS1 scales in or out.

Solution: From Azure Monitor, configure the autoscale settings.

Does this meet the goal?

A. Yes

B. No

ANSWER: A

Explanation:

Yes is correct. Azure Monitor autoscale settings are the appropriate place to configure scaling behavior for an Azure virtual machine scale set and to notify operators when an autoscale action occurs. An autoscale setting supports a notification configuration that can send email when the autoscale engine performs a scale action. This notification can be sent to subscription administrators and optionally to specified email addresses, allowing the team to receive an email for both scale-out and scale-in activity on VMSS1.

The requirement concerns an operational event in the Azure-hosted compute resource, rather than a build or deployment event in Azure DevOps. Therefore, configuring the notification as part of the VMSS autoscale setting directly associates the alert with the event that must be monitored: the autoscale action itself. Azure Monitor evaluates the configured rules, applies the capacity change when the rule conditions are met, and sends the configured notification for that action. This fulfills the stated goal without requiring changes to Project1, the App1 pipeline, or VMSS deployment configuration. Microsoft documents autoscale configuration and its email and webhook notification actions in Get started with autoscale in Azure Monitor and Use autoscale actions to send email and webhook alert notifications.

QUESTION NO: 56 - (SIMULATION)

SIMULATION

You have an Azure functionhosted in an App Service plan named az400-9940427-func1.

You need to configure az400-9940427-func1 to upgrade the functions automatically whenever new code is committed to the master branch of https://github.com/Azure-Samples/functions-quickstart.

To complete this task, sign in to the Microsoft Azure portal.

ANSWER: See the explanation for the answer

Explanation:

Configure continuous deployment from the GitHub repository by using Azure Pipelines:

1. In the Azure portal, open the Function App `az400-9940427-func1`.
2. Select **Deployment Center**.
3. For the build provider, select **Azure Pipelines (Preview)** and select **Continue**.
4. In the code configuration, select **GitHub** as the source and authorize GitHub if prompted.
5. Select the following GitHub values:
   Organization: `Azure-Samples`
   Repository: `functions-quickstart`
   Branch: `master`
6. Select **Continue** through the optional test and staging/deployment-slot pages, choosing the desired defaults (they are not required for the requested CI trigger).
7. Review the summary and select **Finish**.

This creates/configures an Azure Pipeline with a continuous-integration trigger for the repository's `master` branch. New commits to `master` will build and deploy the function app automatically.

QUESTION NO: 57

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You have an approval process that contains a condition. The condition requires that releases be approved by a team leader before they are deployed.

You have a policy stating that approvals must occur within eight hours.

You discover that deployment fail if the approvals take longer than two hours.

You need to ensure that the deployments only fail if the approvals take longer than eight hours.

Solution: From Post-deployment conditions, you modify the Time between re-evaluation of gates option.

Does this meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. The requirement is to allow a team leader's required pre-deployment approval to remain pending for as long as eight hours, failing the deployment only when that approval has not been granted within that period. In a classic Azure DevOps release pipeline, this behavior is controlled by the approval timeout configured for the pre-deployment approval on the environment or stage. The timeout defines how long Azure DevOps waits for the required manual approval before rejecting or failing the pending deployment.

“Time between re-evaluation of gates” has a different purpose. Gates are automated validation checks, such as querying work items, invoking an Azure Function, or checking an external monitoring system. This setting controls the interval at which Azure DevOps reruns those automated gate evaluations while it is waiting for gates to succeed. It does not control the duration for which a manual approver can leave a pre-deployment approval pending. Furthermore, the proposed setting is under post-deployment conditions, whereas the stated approval is required before deployment begins. Therefore, modifying this gate re-evaluation interval cannot extend the approval deadline from two hours to eight hours. Microsoft distinguishes manual approvals from automated gates and documents approval timeout as an approval-specific setting. See Azure DevOps approvals and gates and Configure gates for classic release pipelines.

QUESTION NO: 58

You have a project in Azure DevOps.

You plan to deploy a self-hosted agent by using an unattended configuration script.

Which two values should you define in the configuration script? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. authorization credentials

B. the project name

C. the deployment group name

D. the organization URL

E. the agent pool name

ANSWER: A D

Explanation:

An unattended self-hosted-agent configuration must provide authorization credentials and the organization URL. The organization URL identifies the Azure DevOps Services organization to which the agent will connect and register, typically in the form `https://dev.azure.com/organization`. In the agent configuration command, this is supplied through the `--url` parameter. Authorization credentials are also required because the agent must authenticate before Azure DevOps can register it and allow it to receive jobs. A personal access token is commonly used for this purpose, supplied with `--auth pat` and `--token`. The token must have appropriate Agent Pools permissions, such as the ability to read and manage the target pool. Microsoft documents unattended setup using `config.cmd --unattended --url ... --auth pat --token ...`, with the URL and authentication details replacing prompts that would otherwise be entered interactively. The pool can be specified when a nondefault destination is needed, but the essential values identified here establish the service endpoint and authenticated registration context. See Configure a self-hosted Windows agent unattended and Personal access tokens for agent registration.

QUESTION NO: 59

You have an Azure DevOps organization.

You are designing an automated security and compliance solution that will use GitHub Advanced Security for Azure DevOps.

You need to ensure that security checks and compliance requirements are enforced automatically throughout the development lifecycle.

Which two features should you include in the solution? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. automated approvals

B. code linting

C. secret scanning push protection

D. dependency scanning

E. pull request annotation scanning

ANSWER: C D

Explanation:

secret scanning push protection and **dependency scanning** provide automated controls at key points in the development lifecycle through GitHub Advanced Security for Azure DevOps. Secret scanning push protection evaluates commits as developers push them and prevents supported secret patterns from being introduced into the repository. This shifts credential protection earlier than a post-commit review and helps organizations enforce a policy that sensitive credentials must not be committed to source control. Administrators can configure the feature and review bypass events, supporting compliance visibility and governance.

Dependency scanning continuously identifies open-source dependencies with known vulnerabilities and creates alerts for affected packages. Development teams can review vulnerability information, affected dependency paths, severity, and available remediation guidance directly in their development workflow. This enables ongoing monitoring rather than relying on a one-time assessment, which is essential because new vulnerabilities can be disclosed after code has already been committed or deployed. Together, these capabilities automatically protect source code from exposed secrets and identify supply-chain risks in application dependencies. See Secret scanning for Azure DevOps and Dependency scanning for Azure DevOps.

QUESTION NO: 60

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You plan to update the Azure DevOps strategy of your company.

You need to identify the following issues as they occur during the company's development process:

Solution: You implement continuous integration.

Does this meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. Continuous integration is a development practice in which developers frequently merge code changes into a shared repository and an automated build and test process validates those changes. Azure Pipelines supports this model by triggering pipeline runs when code is committed or a pull request is updated. Therefore, continuous integration provides an important automation point for executing quality, security, dependency, and compliance checks early and repeatedly.

However, implementing continuous integration by itself does not provide the specialized analysis required to identify dependency-related development issues, such as vulnerable open-source components, license obligations, or use of prohibited packages. Those findings require a software composition analysis, dependency-scanning, or license-governance capability to be configured as part of the engineering workflow. The CI process can run that capability automatically, but CI is not the capability that discovers and evaluates component risk. Consequently, the stated solution alone does not meet the goal; the pipeline must include an appropriate scanning tool and policy evaluation. Microsoft describes Azure Pipelines as an automated CI/CD service, and Microsoft Defender for DevOps provides security visibility and recommendations for DevOps environments. See What is Azure Pipelines? and Microsoft Defender for DevOps.

QUESTION NO: 61

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You integrate a cloud-hosted Jenkins server and a new Azure DevOps deployment.

You need Azure DevOps to send a notification to Jenkins when a developer commits changes to a branch in Azure Repos.

Solution: You add a trigger to the build pipeline.

Does this meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. The stated goal is for Azure DevOps to notify an external, cloud-hosted Jenkins server when a commit is made to a branch in Azure Repos. Azure Pipelines build triggers respond to repository changes by queuing or running an Azure Pipelines build; they do not constitute an outbound notification mechanism for a Jenkins server. To send an event from Azure DevOps to Jenkins, configure an Azure DevOps service hook subscription. Service hooks publish Azure DevOps events to external services, and the Jenkins service hook is designed to trigger a configured Jenkins job from project events. For this requirement, select the appropriate Azure Repos event, such as a code push event, apply branch filters as needed, and configure the Jenkins endpoint and job details. This gives Jenkins the notification it needs to initiate its own build or other automation after code is committed. Microsoft documents Jenkins as a supported Azure DevOps service hook consumer and describes service hooks as the integration feature for sending project-event notifications to external services. See Jenkins service hooks for Azure DevOps and Azure DevOps service hooks overview.

QUESTION NO: 62

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

Your company has a project in Azure DevOps for a new web application.

You need to ensure that when code is checked in, a build runs automatically.

Solution: From the Triggers tab of the build pipeline, you select Enable continuous integration.

Does this meet the goal?

A. Yes

B. No

ANSWER: A

Explanation:

Yes. Enabling continuous integration from the Triggers tab of a classic Azure DevOps build pipeline configures the pipeline to queue builds automatically when qualifying source-code changes are pushed to the repository. This directly implements

the stated requirement that a build run when code is checked in. The trigger can also be scoped to particular branches, paths, or repositories, allowing the team to ensure that only relevant changes start a build. For the baseline requirement, however, selecting *Enable continuous integration* activates the appropriate automated build behavior. Continuous integration provides prompt validation of incoming changes by building the updated source automatically, which helps identify integration and build issues close to the time of check-in. Microsoft documents CI triggers as the mechanism for automatically running pipelines after repository updates, and the classic pipeline editor exposes this capability through its Triggers tab. See Azure Pipelines build triggers and CI builds for Git repositories in Azure Pipelines.

QUESTION NO: 63

You use GitHub for source control and Azure Boards for project management. GitHub and Azure Boards are integrated.

You plan to create a pull request in GitHub.

You need to automatically link the request to an existing Azure Boards work item by using the text of AB#.

To which two elements can you add the text? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. milestone

B. comment

C. title

D. description

E. label

ANSWER: C D

Explanation:

The correct locations are **title** and **description**. With the Azure Boards and GitHub integration enabled, Azure Boards recognizes the `AB#<workitemnumber>` syntax in a GitHub pull request title or pull request description. For example, including `AB#123` in either field creates an association between that pull request and Azure Boards work item 123. The linked development artifact is then visible from the work item, providing traceability from planned work to the code review and eventual code change. This linkage is useful in delivery workflows because reviewers and project stakeholders can navigate directly between the pull request and the related backlog item, bug, task, or user story. A title can contain the reference when the pull request primarily addresses one work item. A description can contain the reference alongside implementation context, test information, and other pull-request details. Microsoft documents that GitHub pull-request titles and descriptions support Azure Boards work-item mentions using the `AB#ID` format when the connection between Azure Boards and GitHub has been configured. See Link GitHub commits, pull requests, and issues to work items and Connect Azure Boards to GitHub.

QUESTION NO: 64

You have created an Azure DevOps project for a new application that will be deployed to a number of Windows Server 2016 Azure virtual machines.

You are preparing a deployment solution that allows for the virtual machines to maintain a uniform configuration, and also keep administrative effort with regards to configuring the virtual machines to a minimum.

Which of the following should be part of your solution? (Choose two.)

A. Azure Resource Manager templates

B. The PowerShell Desired State Configuration (DSC) extension for Windows

C. Azure pipeline deployment groups

D. The Custom Script Extension for Windows

E. Azure pipeline stage templates

ANSWER: A B

Explanation:

Azure Resource Manager templates and The PowerShell Desired State Configuration (DSC) extension for Windows provide an infrastructure-as-code and configuration-as-code approach for consistently deploying and managing Windows Server virtual machines. Azure Resource Manager templates declare the Azure resources to deploy, including virtual machines, networking, managed disks, and VM extensions. Because the template is version-controlled and reusable, the same infrastructure definition can be deployed repeatedly across environments without relying on manual portal configuration.

The PowerShell Desired State Configuration (DSC) extension for Windows applies and maintains the intended operating-system configuration inside each Windows VM. A DSC configuration can define required Windows features, services, files, registry settings, and application prerequisites. Applying that configuration through the extension makes server setup repeatable and enables machines to converge on the declared configuration, reducing ongoing administrative work and configuration drift. The template can deploy each VM and associate its DSC extension as part of one automated deployment workflow. This directly supports the requirement for uniform VM configuration while minimizing manual server administration. Microsoft documents declarative resource deployment through Azure Resource Manager templates and configuration of Windows VMs through the PowerShell DSC extension.

QUESTION NO: 65

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You plan to create a release pipeline that will deploy Azure resources by using Azure Resource Manager templates. The release pipeline will create the following resources:

- Two resource groups
- Four Azure virtual machines in one resource group
- Two Azure SQL databases in other resource group

You need to recommend a solution to deploy the resources.

Solution: Create a main template that has two linked templates, each of which will deploy the resources in its respective group.

Does this meet the goal?

A. Yes

B. No

ANSWER: A

Explanation:

Yes is correct. Azure Resource Manager templates support modularizing a deployment by using a main template that invokes linked templates. This pattern allows the release pipeline to coordinate the complete environment deployment while keeping the definitions for distinct workloads separate and maintainable. In this case, the overall deployment can be executed at subscription scope, which is required when creating resource groups. The main template can create the two resource groups and invoke deployments targeted at each group. One linked template can define the virtual machines and their dependent resources for the designated resource group, while the other linked template can define the Azure SQL databases in the other group.

Using linked templates is appropriate when a solution contains logically separate resource sets, when templates should be independently maintained, or when a single template would otherwise become difficult to manage. Deployment dependencies can also be defined so that resource-group creation completes before the group-scoped resource deployments begin. Azure Resource Manager supports linked templates through the `templateLink` property of a deployment resource, and subscription-level deployments can deploy to resource groups within the subscription. Microsoft documents these capabilities in its guidance for linked and nested templates and subscription-scope deployments.

QUESTION NO: 66

You are deploying a server application that will run on a Server Core installation of Windows Server 2019.

You create an Azure key vault and a secret.

You need to use the key vault to secure API secrets for third-party integrations.

Which three actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

D18912E1457D5D1DDCBD40AB3BF70D5D

A. Configure RBAC for the key vault.

B. Modify the application to access the key vault.

C. Configure a Key Vault access policy.

D. Deploy an Azure Desired State Configuration (DSC) extension.

E. Deploy a virtual machine that uses a system-assigned managed identity.

ANSWER: B C E

Explanation:

Modify the application to access the key vault is required because the application must request the API secret at runtime through Azure Key Vault rather than retaining the third-party credential in application code, local configuration, or deployment artifacts. The application can use an Azure SDK client or the Key Vault REST API to retrieve the specific secret it requires.

Deploy a virtual machine that uses a system-assigned managed identity gives the workload an automatically managed Microsoft Entra ID identity. Azure manages the identity lifecycle, and the application can obtain an access token for Key Vault without storing an Azure client secret, certificate, or password on the Server Core host.

Configure a Key Vault access policy grants that managed identity the necessary data-plane secret permission, such as *Get*, following least-privilege principles. With these elements in place, the VM-hosted application authenticates by using its managed identity and is authorized to retrieve the protected API secret directly from the vault. Microsoft documents this managed-identity pattern for virtual machines at Use Key Vault from an Azure virtual machine and describes Key Vault secret authorization models at Provide access to Key Vault keys, certificates, and secrets with an Azure role-based access control.

QUESTION NO: 67 - (DRAG DROP)

DRAG DROP

Your company plans to deploy an application to the following endpoints:

- Ten virtual machines hosted in Azure
- Ten virtual machines hosted in an on-premises data center environment

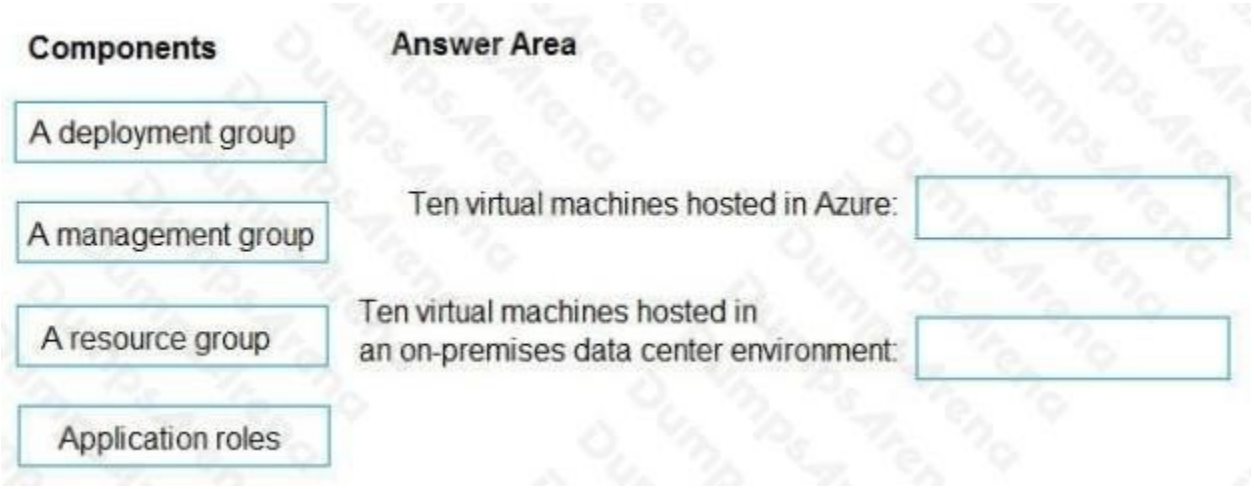
All the virtual machines have the Azure Pipelines agent.

You need to implement a release strategy for deploying the application to the endpoints.

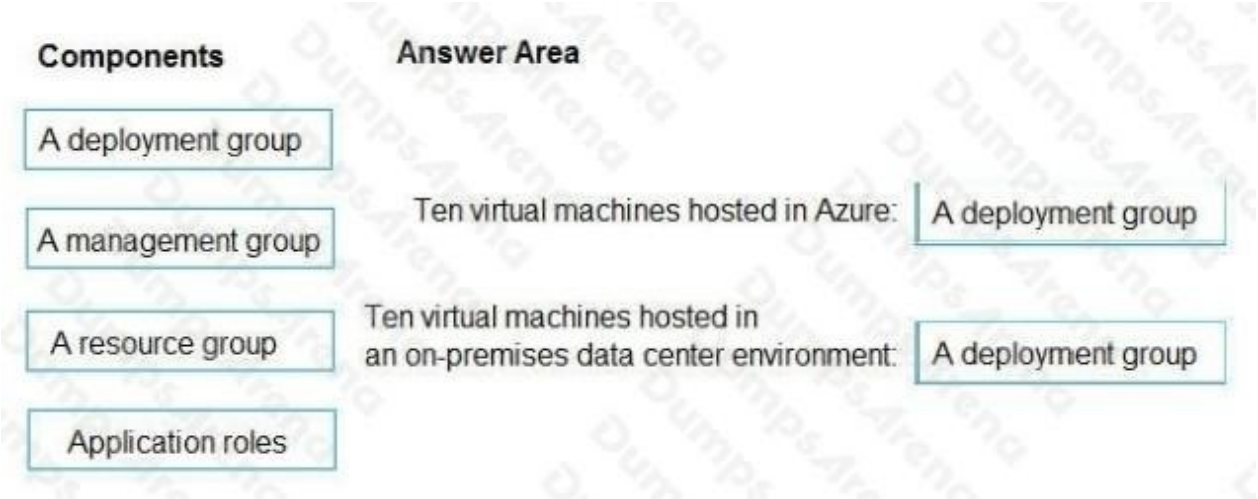
What should you recommend using to deploy the application to the endpoints? To answer, drag the appropriate components to the correct endpoints. Each component may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content.

NOTE: Each correct selection is worth one point.

Select and Place:



ANSWER:



Explanation:

A deployment group is the appropriate component for both sets of virtual machines. A deployment group is an Azure DevOps collection of deployment target machines that run the Azure Pipelines agent. A release pipeline can use that group to execute deployment tasks directly on every registered target, including deploying in parallel, in batches, or according to a configured deployment strategy.

The deciding fact is that all twenty virtual machines already have the Azure Pipelines agent. Each agent can register its machine with Azure DevOps as part of a deployment group, giving the release pipeline a direct and consistent way to run deployment steps on those machines. This agent-based approach is not tied to a particular hosting location. The target machines can be Azure virtual machines, servers in an on-premises datacenter, or machines in another connected environment, provided that the agents can communicate with Azure DevOps.

Therefore, one deployment group can represent the Azure-hosted machine targets and another deployment group can represent the on-premises machine targets, or the same component type can be used wherever the release design requires target-machine deployment. Azure DevOps classic release pipelines specifically support deployment groups for deploying applications to groups of physical or virtual machines. Microsoft documents this deployment-target model in Deployment

groups in Azure Pipelines. Details about installing, configuring, and using the agent on self-hosted machines are available in Azure Pipelines agents.

QUESTION NO: 68

You need to execute inline testing of an Azure DevOps pipeline that uses a Docker deployment model. The solution must prevent the results from being published to the pipeline.

What should you use for the inline testing?

A. a single stage Dockerfile

B. an Azure Kubernetes Service (AKS) pod

C. a multi-stage Dockerfile

D. a Docker Compose file

ANSWER: C

Explanation:

A multi-stage Dockerfile is the appropriate approach because it can define separate build, test, and runtime stages in one container build. The pipeline invokes the Docker build, and the test commands run within an intermediate stage. Docker only produces the stage specified as the final target (or the last stage by default) as the deployable image, so test tooling and test-stage artifacts do not need to be included in the runtime image. This enables tests to act as a build gate: if a test command exits with a nonzero status, the Docker build fails and the pipeline stops before deployment. At the same time, test results are not automatically made available as Azure Pipelines test results. Publishing requires an explicit pipeline action, such as using the PublishTestResults task and making result files accessible to that task. Keeping the test execution inside the intermediate Docker stage without copying result files out or adding a publishing task satisfies the requirement for inline tests whose results are not published to the pipeline. Microsoft describes multi-stage builds as a way to use multiple build environments while retaining only the artifacts required by the final image. See Docker's multi-stage build documentation and Microsoft guidance for building container images in Azure Pipelines.

QUESTION NO: 69

You have a project in Azure DevOps.

You plan to deploy a self-hosted agent by using an unattended configuration script.

Which two values should you define in the configuration script? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth onepoint.

A. authorization credentials

B. the project name

C. the deployment group name

D. the organization URL

E. the agent pool name

ANSWER: A D

Explanation:

An unattended Azure Pipelines agent configuration must identify the Azure DevOps organization to which the agent will be registered and must authenticate the registration operation. Therefore, **the organization URL** and **authorization credentials** are required values. The organization URL is supplied through the agent configuration command's `--url`

parameter, typically in the form `https://dev.azure.com/organization-name`. This establishes the Azure DevOps service endpoint that receives the agent registration.

Authorization credentials allow the configuration process to prove that it is authorized to add an agent. For Azure DevOps Services, unattended setup commonly uses `--auth pat` together with `--token` containing a personal access token that has the appropriate Agent Pools permissions. The authenticated identity is used to register and manage the agent connection. Microsoft documents that unattended configuration requires both the server URL and credentials for an identity authorized to configure agents. Configuration can additionally specify settings such as an agent pool, agent name, and work directory, but the service URL and authentication information establish the essential connection and authorization needed to complete registration. See Unattended agent configuration and PAT-based agent registration.

QUESTION NO: 70

Your company is concerned that when developers introduce open source libraries, it creates licensing compliance issues.

You need to add an automated process to the build pipeline to detect when common open source libraries are added to the code base.

What should you use?

A. SourceGear Vault

B. Jenkins

C. Microsoft Visual SourceSafe

D. WhiteSource Bolt

ANSWER: D

Explanation:

WhiteSource Bolt is the appropriate choice because it provides software composition analysis for open-source dependencies in Azure DevOps. When integrated into an Azure Pipelines build, it scans the project and its dependency manifests to identify open-source components introduced into the codebase. It then produces an inventory of those components and evaluates associated license information, helping teams identify licensing-compliance concerns early in the continuous integration process. This automated feedback is particularly useful because developers can discover dependency and license implications as part of normal builds rather than through a separate, manual audit later in the release cycle. WhiteSource Bolt was the Azure DevOps extension for this purpose; the product is now associated with Mend, but the named extension remains the answer that matches the question. Microsoft's Azure DevOps Labs walkthrough describes configuring WhiteSource Bolt in a pipeline and reviewing open-source security and license results. The Visual Studio Marketplace listing also documents the extension's dependency-analysis role for Azure DevOps projects. See Azure DevOps Labs: WhiteSource Bolt and WhiteSource Bolt on Visual Studio Marketplace.

QUESTION NO: 71 - (HOTSPOT)

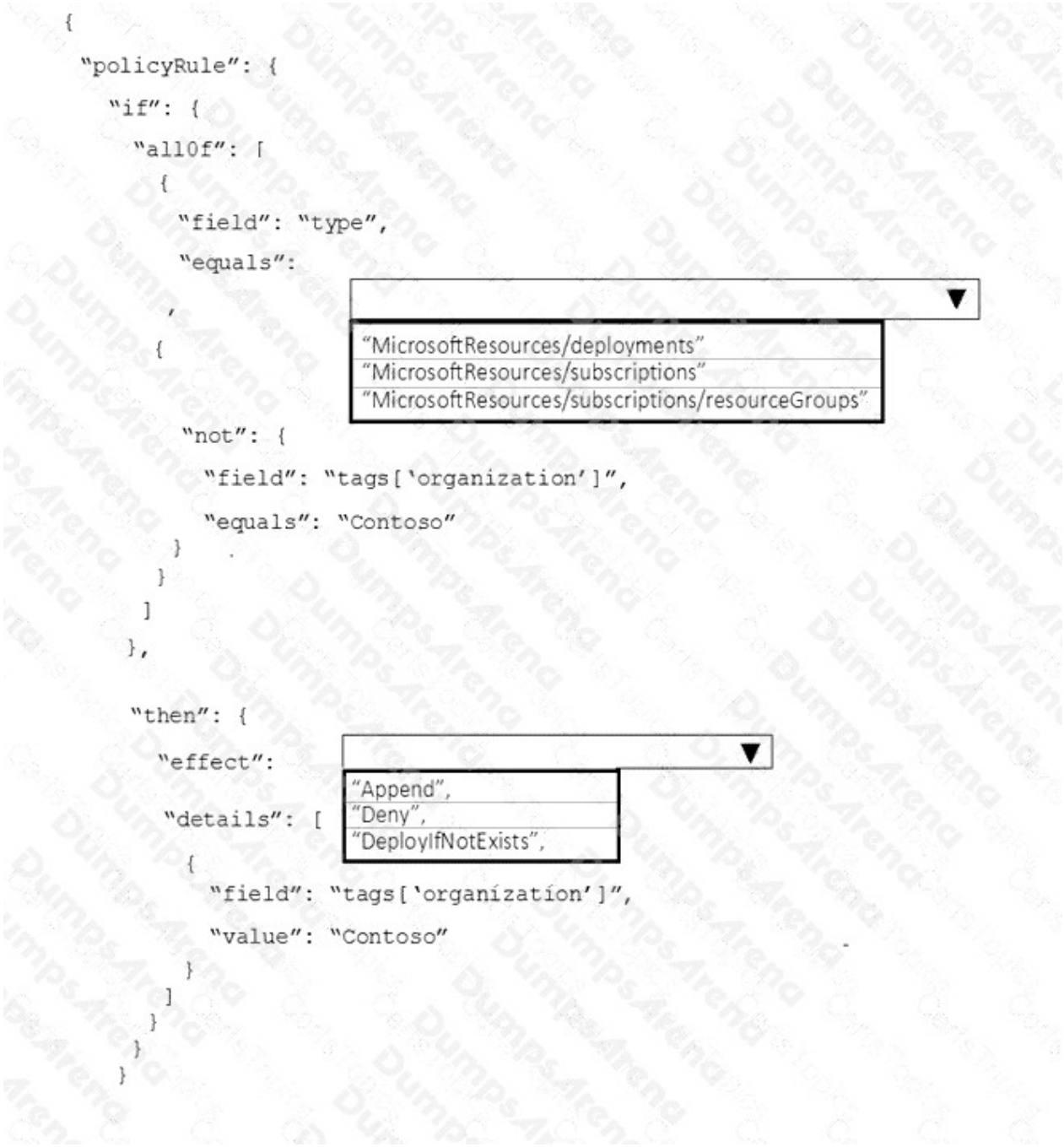
Your company has an Azure subscription.

The company requires that all resource group in the subscription have a tag named organization set to a value of Contoso.

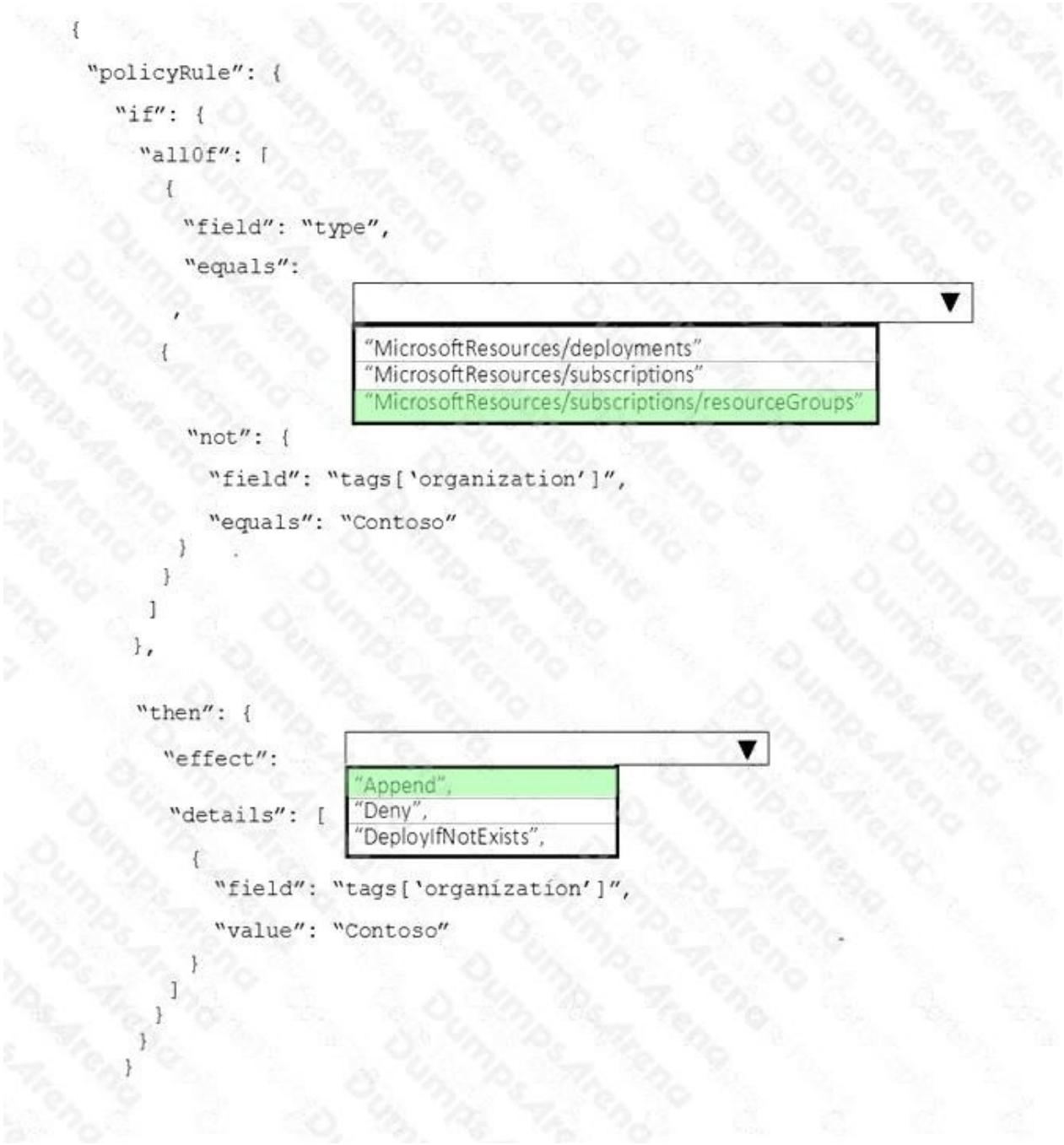
You need to implement a policy to meet the tagging requirement.

How should you complete the policy? To answer, select the appropriate options in the answer area.

NOTE: Each correct selection is worth one point.


{
"policyRule": {
"if": {
"allOf": [
{
"field": "type",
"equals":
,
{
"not": {
"field": "tags['organization']",
"equals": "Contoso"
}
}
]
},
"then": {
"effect":
"details": [
{
"field": "tags['organization']",
"value": "Contoso"
}
]
}
}
}
"MicrosoftResources/deployments"
"MicrosoftResources/subscriptions"
"MicrosoftResources/subscriptions/resourceGroups"
"Append",
"Deny",
"DeployIfNotExists",

ANSWER:



Explanation:

The policy condition must target resource groups, so the resource type is `Microsoft.Resources/subscriptions/resourceGroups`. Azure Policy evaluates the `type` alias for the resource being created or updated. Restricting the rule to this resource type ensures that the tag requirement is applied to resource groups in the subscription rather than to unrelated Azure resources.

The rule then checks whether `tags['organization']` does not equal `Contoso`. When that condition is true, the required effect is `Append`. Append modifies the request content during policy evaluation by appending the configured field and value. In this case, the `details` section supplies the exact tag field, `tags['organization']`, and the required value, `Contoso`. This makes the resulting resource-group request include the organization tag with the mandated value.

This structure is specifically consistent with Azure Policy's append-effect syntax: an effect declaration is paired with a details collection containing the resource property to append and its value. The policy can therefore enforce the organization metadata convention as resource groups are created or updated, without requiring each deployment author to add the tag manually. Microsoft documents the Append effect and its field/value details format at Azure Policy append effect. Azure resource-group tags and their use for organizing resources are also documented at Use tags to organize your Azure resources and management hierarchy.

QUESTION NO: 72

Your company uses cloud-hosted Jenkins for builds.

You need to ensure that Jenkins can retrieve source code from Azure Repos.

Which three actions should you perform? Each correct answer presents part of the solution

NOTE: Each correct answer selection is worth one point

A. Add the Team Foundation Server (TFS) plug-in to Jenkins.

B. Create a personal access token in your Azure DevOps account.

C. Create a webhook in Jenkins.

D. Add a domain to your Jenkins account.

E. Create a service hook in Azure DevOps.

ANSWER: A B E

Explanation:

Installing the Team Foundation Server (TFS) plug-in in Jenkins provides the Azure DevOps/TFS integration required for Jenkins jobs to work with Azure Repos. The plug-in enables a Jenkins job to identify an Azure DevOps collection or organization, project, repository, and branch, so Jenkins can obtain the source used by the build. Microsoft documents this Jenkins integration as using the Team Foundation Server plug-in for Azure DevOps repository support.

Creating a personal access token in the Azure DevOps account supplies credentials that Jenkins can use to authenticate to Azure DevOps. The token should be granted the minimum required permissions, including repository Code read access, and then stored as a Jenkins credential rather than placed directly in a job definition. This allows the source checkout to occur securely in a cloud-hosted Jenkins environment.

Creating a service hook in Azure DevOps completes the automated integration by notifying Jenkins when relevant repository events occur, such as a code push. The Azure DevOps Jenkins service hook can trigger the configured Jenkins build, after which the job uses its Azure Repos configuration and credentials to retrieve the applicable revision. See Jenkins service hooks in Azure DevOps and the Team Foundation Server Jenkins plug-in documentation.

QUESTION NO: 73

After you answer a question in this section, you will NOT be able to return to it As a result, these questions will not appear in the review screen.

You use Azure Pipelines to build and test a React js application

You have a pipeline that has a single job.

You discover that installing JavaScript packages from npm lakes approximately five minutes each time you run the pipeline.

You need to recommend a solution to reduce the pipeline execution time.

Solution: You recommend enabling pipeline caching.

Does this meet the goal?

A. Yes

B. No

ANSWER: A

Explanation:

Yes is correct. Azure Pipelines caching is intended to reduce repeated build work by restoring files from prior pipeline runs rather than downloading or regenerating them each time. In a React.js pipeline, downloading npm packages can consume substantial time, particularly when the dependency set remains unchanged between runs. A pipeline cache can persist npm's package cache directory and restore it at the start of later runs. The cache key should incorporate the dependency lock file, such as `package-lock.json`, so the cache is reused only when the declared dependency graph is unchanged. When a dependency update changes the lock file, Azure Pipelines creates and uses an appropriate new cache entry. This reduces the network and package-download overhead while preserving deterministic package installation through commands such as `npm ci`. Microsoft's JavaScript pipeline guidance includes npm caching patterns, and its pipeline caching documentation identifies dependency caching as a primary use case for improving build performance. Therefore, enabling pipeline caching directly addresses the repeated five-minute npm installation cost and meets the goal of reducing execution time for subsequent pipeline runs. See Pipeline caching in Azure Pipelines and Build JavaScript apps with Azure Pipelines.

QUESTION NO: 74

You are designing a YAML template for use with Azure Pipelines. The template Will include theOutputfile parameter.

Which two methods can you use to reference the parameter? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. Option A

B. Option B

C. ${{ parameters.Outputfile }}

D. ${{ parameters['Outputfile'] }}

E. Option E

ANSWER: C D

Explanation:

Azure Pipelines template parameters are evaluated during template parsing and expansion, before pipeline execution. The `${{ parameters.Outputfile }}` expression uses property dereference syntax to retrieve the value of the `Outputfile` parameter. This is the concise form commonly used when the parameter name is a valid identifier. The `${{ parameters['Outputfile'] }}` expression uses index syntax and retrieves the same parameter value. Index syntax is particularly useful for parameter names that cannot be conveniently expressed with property syntax, such as names containing characters that are not valid in an identifier. Both forms are compile-time template expressions, so Azure Pipelines substitutes the supplied parameter value while expanding the YAML template. For example, either expression can be placed in a task input, script argument, variable value, or conditional template construct where template expressions are supported. This behavior is distinct from runtime macro variables, which use `$(name)` syntax and are resolved later during job or task execution. Microsoft documents both property and index dereference forms for accessing values in Azure Pipelines expressions. See Azure Pipelines expressions and Template parameters.

QUESTION NO: 75

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You plan to update the Azure DevOps strategy of your company.

You need to identify the following issues as they occur during the company's development process: ▪ Licensing violations

▪ Prohibited libraries

Solution: You implement continuous deployment.

Does this meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. Continuous deployment automates the delivery of a validated build to one or more environments. Its central purpose is to make releases repeatable, fast, and reliable after the software has passed the applicable build and validation processes. Implementing continuous deployment alone does not provide a capability to discover third-party component licenses or determine whether a dependency is on an organization's prohibited-library list.

Detecting licensing violations and restricted open-source libraries requires software composition analysis and dependency-policy checks. These checks should be incorporated into the development workflow, commonly as part of continuous integration builds or pull-request validation, so that every code or dependency change is evaluated before it progresses toward deployment. Azure DevOps pipelines can run security and compliance tooling, and the resulting findings can be used as quality gates to prevent noncompliant builds from proceeding. Microsoft's Microsoft Security DevOps extension is designed to integrate security analysis tools into Azure DevOps pipelines; dependency and license governance can similarly be enforced by an appropriate software composition analysis tool.

Microsoft distinguishes continuous integration, which automatically builds and tests code changes, from continuous deployment, which deploys validated changes. The required detection capability belongs in the validation process rather than being supplied by deployment automation itself. See Azure Pipelines key concepts and Microsoft Security DevOps for Azure DevOps.

QUESTION NO: 76

You use GitHub for source control

You are evaluating whether to use proxying to add a private upstream MyGet package feed to your MyGet feed. What are two possible advantages of this approach? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.

A. minimizes latency when accessing the package

B. provides automatic authentication

C. minimizes the impact on your storage quota

D. minimizes the impact of upstream source availability issues

ANSWER: B C

Explanation:

Using a MyGet package-source proxy lets the MyGet feed serve as a controlled access point for packages hosted by a separate, private upstream feed. "provides automatic authentication" is an advantage because credentials for the private upstream source can be configured and managed at the MyGet feed level. Developers and build agents can then use the MyGet endpoint rather than requiring separate direct authentication configuration for the upstream feed. This centralizes access management and is particularly useful for automated builds, where secrets should not be distributed broadly across developer workstations and pipeline configurations.

"minimizes the impact on your storage quota" is also an advantage because proxying makes upstream packages available through the MyGet feed without requiring the packages to be published as packages hosted directly in that feed. The proxy provides a unified consumption endpoint while leaving package ownership and primary storage with the upstream source. This can be useful when teams need access to dependencies from private external feeds but do not want to copy or maintain duplicate package content in their own feed. MyGet's package-source documentation describes configuring

external package sources and proxy behavior: MyGet package sources. For the related feed-aggregation pattern, see Azure Artifacts upstream sources.

QUESTION NO: 77

You have a project in Azure DevOps named Project1. Project1 contains a build pipeline named Pipe1 that builds an application named Appl.

You have an agent pool named Pool1 that contains a Windows Server 2019-based self-hosted agent. Pipe1 uses Pool1.

You plan to implement another project named Project2. Project2 will have a build pipeline named Pipe2 that builds an application named App2.

App1 and App2 have conflicting dependencies.

You need to minimize the possibility that the two build pipelines will conflict with each other. The solution must minimize infrastructure costs.

What should you do?

A. Create two container jobs.

B. Change the self-hosted agent to use Red Hat Enterprise Linux (RHEL) 8.

C. Add another self-hosted agent

D. Add a Docker Compose task to the build pipelines.

ANSWER: A

Explanation:

Create two container jobs. A container job runs the pipeline steps in a specified container image while using the existing self-hosted agent to orchestrate the job. Each pipeline can therefore use its own image containing the precise SDK versions, runtimes, package tools, and other dependencies required by its application. This separates the build environments even when both pipelines use the same agent pool and host, substantially reducing the chance that a dependency installed or required for one build affects the other build.

This approach also minimizes infrastructure cost because it reuses the current Windows Server 2019 self-hosted agent rather than requiring another VM or dedicated agent machine. The required build environments are defined as images and can be recreated consistently for each run, which improves repeatability as well as isolation. The host must be configured to run compatible Docker containers, and the selected Windows container images must be compatible with the Windows Server host version. Azure Pipelines supports defining a container at the job level so that the job's steps execute in that containerized environment. See Microsoft Learn: Container jobs in Azure Pipelines and Microsoft Learn: Use Docker with self-hosted agents.

QUESTION NO: 78

You have a widget named Appl and an Azure DevOps YAML pipeline named Pipelinel. App1 is released by using Pipelined Pipeline1 contains a single stage and a single job. You need to ensure that Appl is approved before the app is released. Which two actions should you perform? Each correct answer presents part of the solution. NOTE; Each correct selection is worth one point.

A. Add a stage named deployment to the YAML pipeline.

B. Change the existing job in Pipeline1 to a deployment job.

C. Create a service connection.

D. Create an environment and add an approval check.

E. Configure branch control.

ANSWER: B D

Explanation:

Changing the existing job in Pipeline1 to a deployment job enables the pipeline to target an Azure DevOps environment. A deployment job is the YAML job type designed for application deployments, and its `environment` property associates a deployment with a named environment. This association provides deployment history and, crucially, makes the environment a protected resource that can enforce checks before deployment starts.

Creating an environment and adding an approval check supplies the required approval gate. Environment owners can configure an Approval check and specify one or more approvers. When the deployment job attempts to use that environment, Azure DevOps pauses execution until the configured approver approves the check. The approval configuration is managed outside the YAML file, which prevents pipeline authors from bypassing the required authorization merely by editing pipeline code. Together, an environment-level approval check and a deployment job that consumes the environment ensure approval is obtained before the release proceeds. For implementation details, see Azure Pipelines approvals and checks and Azure Pipelines deployment jobs.

QUESTION NO: 79

You use GitHub Actions to run code scanning for a JavaScript application.

You need to create a workflow that performs CodeQL analysis and publishes the findings to the repository security tab.

Which two CodeQL actions should you include? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. github/codeql-action/init

B. github/codeql-action/analyze

C. actions/upload-artifact

D. github/dependency-review-action

E. actions/cache

ANSWER: A B

Explanation:

Include **github/codeql-action/init** to initialize CodeQL analysis and specify the language to scan. For this scenario, the language can be configured as `javascript-typescript`. The initialization action prepares the CodeQL database and analysis configuration for the workflow.

Include **github/codeql-action/analyze** after the application build steps. The analyze action runs the CodeQL queries and uploads the resulting SARIF findings to GitHub code scanning. The findings are then available in the repository security views and can be used by code-scanning alerts and security policies. See Configuring CodeQL code scanning in your workflow.

QUESTION NO: 80

You have an Azure Key Vault named KV1 and an Azure DevOps project named Project1.

You need to make selected secrets from KV1 available to a pipeline as variables. The secrets must remain centrally managed in Key Vault so that new secret versions can be consumed without updating the pipeline definition.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Create a variable group linked to Azure Key Vault.

B. Assign the Key Vault Secrets User role to the service connection identity.

C. Store the secret values as plain-text YAML variables.

D. Assign the Owner role on the Azure subscription to the service connection identity.

E. Publish the secrets as a pipeline artifact.

ANSWER: A B

Explanation:

Create a **variable group linked to Azure Key Vault**. A linked variable group maps selected Key Vault secrets to pipeline variables. The secret values remain in Key Vault, and Azure Pipelines retrieves them when the pipeline runs. This allows a new version of a selected secret to be consumed without hard-coding the value in YAML or manually changing the pipeline variable value.

Grant the identity used by the Azure Resource Manager service connection the **Key Vault Secrets User** role on KV1 when the vault uses the Azure RBAC permission model. This role grants the data-plane permission needed to read secret values while following least-privilege principles. See Link secrets from an Azure key vault and Provide access to Key Vault keys, certificates, and secrets with Azure RBAC.

QUESTION NO: 81

Your company has an Azure DevOps project,

The source code for the project is stored in an on-premises repository and uses on an on-premises build

server.

You plan to use Azure DevOps to control the build process on the build server by using a self-hosted agent.

You need to implement the self-hosted agent.

You download and install the agent on the build server.

Which two actions should you perform next? Each correct answer presents part of the solution.

A. From Azure, create a shared access signature (SAS).

B. From the build server, create a certificate, and then upload the certificate to Azure Storage.

C. From the build server, create a certificate, and then upload the certificate to Azure Key Vault.

D. From DevOps, create a personal access token (PAT).

E. From the build server, run config.cmd.

ANSWER: D E

Explanation:

To register and use a self-hosted Azure Pipelines agent on the on-premises build server, create a personal access token (PAT) in Azure DevOps and run `config.cmd` on that server. The PAT supplies the authentication needed during initial agent registration. For this scenario, it should be created by an identity authorized to administer the target agent pool; Azure DevOps guidance identifies the Agent Pools read and manage scope as the relevant PAT permission when configuring an agent. The PAT is entered when the configuration process requests credentials, allowing the agent to securely register with the Azure DevOps organization and connect to the selected pool.

Running `config.cmd` from the extracted agent directory starts the interactive configuration workflow. It collects the Azure DevOps organization URL, authentication method and PAT, agent pool, and agent name. The workflow can also configure the agent to run as a Windows service, which enables it to accept pipeline jobs without requiring an interactive user session. Once configuration completes and the agent is running, Azure Pipelines can dispatch build jobs to the on-premises machine;

the agent makes the required outbound connection to Azure DevOps. Microsoft documents the installation and configuration sequence at Self-hosted Windows agents and documents PAT creation and scope management at Use personal access tokens.

QUESTION NO: 82

You use GitHub for source control of a Node.js application.

You need to automatically create pull requests to update dependencies and to remediate vulnerable dependencies.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Create a dependabot.yml configuration file for version updates.

B. Enable Dependabot security updates for the repository.

C. Enable GitHub Pages for the repository.

D. Create a CODEOWNERS file for package-lock.json.

E. Configure a repository webhook.

ANSWER: A B

Explanation:

Create a `dependabot.yml` configuration file to enable Dependabot version updates. The configuration defines the package ecosystem, directory, and schedule that Dependabot uses to check for newer dependency versions and create update pull requests.

Enable Dependabot security updates for the repository. Dependabot security updates use GitHub security advisories and the repository dependency graph to identify vulnerable dependencies and create pull requests that update them to a non-vulnerable version when available. Version updates and security updates address different requirements and can be used together. See Dependabot options reference and About Dependabot security updates.

QUESTION NO: 83

Your company builds a multi-tier web application.

You use Azure DevOps and host the production application on Azure virtual machines.

Your team prepares an Azure Resource Manager template of the virtual machine that you will use to test new features.

You need to create a staging environment in Azure that meets the following requirements:

- Minimizes the cost of Azure hosting
- Provisions the virtual machines automatically
- Uses the custom Azure Resource Manager template to provision the virtual machines

What should you do?

A. In Azure Cloud Shell, run Azure CLI commands to create and delete the new virtual machines in a staging resource group.

B. In Azure DevOps, configure new tasks in the release pipeline to deploy to Azure Cloud Services.

C. From Azure Cloud Shell, run Azure PowerShell commands to create and delete the new virtual machines in a staging resource group.

D. In Azure DevOps, configure new tasks in the release pipeline to create and delete the virtual machines in Azure DevTest Labs.

ANSWER: D

Explanation:

In Azure DevOps, configure new tasks in the release pipeline to create and delete the virtual machines in Azure DevTest Labs. Azure DevTest Labs is designed for development and test workloads that need rapidly provisioned, governed, and temporary infrastructure. It supports cost-management controls such as automatic shutdown, usage quotas, and restrictions on allowed virtual-machine sizes, helping ensure that a staging environment does not consume hosting resources longer or more broadly than necessary.

DevTest Labs also supports reusable lab environments defined through Azure Resource Manager templates. The team can make its custom template available as a lab environment, then use an Azure DevOps release pipeline to provision the environment for staging validation and delete it when validation is complete. This provides repeatable infrastructure-as-code deployment while integrating environment lifecycle operations into the delivery process. Removing the test virtual machines after use is especially important for minimizing ongoing Azure consumption costs. Microsoft documents pipeline integration in Integrate Azure DevTest Labs into Azure Pipelines and explains how to create DevTest Labs environments from ARM templates in Create environments with Azure Resource Manager templates.

QUESTION NO: 84

You have a widget named Appl and an Azure DevOps YAML pipeline named Pipelinel. App1 is released by using Pipelined Pipeline1 contains a single stage and a single job. You need to ensure that Appl is approved before the app is released. Which two actions should you perform? Each correct answer presents part of the solution. NOTE; Each correct selection is worth one point.

A. Add a stage named deployment to the YAML pipeline.

B. Change the Pipeline1 job to a deployment job.

C. Create a service connection.

D. Create an environment and add an approval check.

E. Configure branch control.

ANSWER: B D

Explanation:

To require approval before the application is released, the pipeline must deploy to an Azure DevOps Environment and that environment must have an approval check configured. A deployment job is the YAML job type designed for deployment operations and can specify an `environment` target. When the deployment job runs, Azure Pipelines recognizes the environment as a protected resource and evaluates its configured checks before allowing the deployment to proceed. Therefore, changing the existing pipeline job to a deployment job provides the mechanism to target the environment without requiring an additional stage; a deployment job can exist in the pipeline's current stage. Creating an environment and adding an approval check establishes the approval gate. The approval is configured outside the YAML by the environment/resource owner, which prevents pipeline authors from bypassing it by editing the pipeline definition. Once an approver approves the pending check, Azure Pipelines permits the deployment job to continue and release the application. Microsoft documents deployment-job environment targeting at Deployment jobs and explains environment approvals and checks at Approvals and checks.

QUESTION NO: 85

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You have an approval process that contains a condition. The condition requires that releases be approved by a team leader before they are deployed.

You have a policy stating that approvals must occur within eight hours.

You discover that deployment fail if the approvals take longer than two hours.

You need to ensure that the deployments only fail if the approvals take longer than eight hours.

Solution: From Pre-deployment conditions, you modify the Time between re-evaluation of gates option.

Does this meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. The *Time between re-evaluation of gates* setting applies to release gates, which are automated checks performed before or after deployment, such as querying Azure Monitor, invoking an Azure Function, or checking a work item query. This setting controls the polling interval: how frequently Azure DevOps reruns configured gate checks while it waits for all gates to succeed. It does not control the maximum time that a manual pre-deployment approval can remain pending.

The stated requirement concerns a team leader's manual approval. In a classic release pipeline, the relevant pre-deployment approval configuration is the approval timeout. That value determines the period Azure DevOps waits for an approver's response before the approval times out and the deployment does not proceed. Therefore, changing only the gate re-evaluation interval cannot change a two-hour approval expiration to eight hours and does not meet the goal. Microsoft distinguishes approvals, which pause deployment for a designated user or group decision, from gates, which repeatedly evaluate automated criteria. See Microsoft documentation on release approvals and gates and Microsoft documentation on approvals and checks.

QUESTION NO: 86

You are designing a YAML template for use with Azure Pipelines. The template Will include the Outputfile parameter.

Which two methods can you use to reference the parameter? Each correct answer presents a complete solution.

NOTE: Each correct selection is worth one point.



A. Option A

B. Option B

C. Option C

D. Option D

E. Option E

ANSWER: C D

Explanation:

Azure Pipelines template parameters are available through the `parameters` context while Azure DevOps expands the template. The parameter can therefore be accessed with template-expression syntax using either property dereference notation, `${{ parameters.Outputfile }}`, or index notation, `${{ parameters['Outputfile'] }}`. Both forms are complete parameter references and are evaluated during template parsing, before runtime execution begins. This is important because template parameters control the template's structure and values at compile time, rather than being resolved as runtime variables during a job or step. Property syntax is convenient when the parameter name is a valid identifier, while index syntax is also supported and is especially useful when accessing names that require quoting. In this case, `Outputfile` can be referenced successfully by either method. Microsoft's template-expression guidance identifies the `parameters` context as the mechanism for accessing template parameters. Its expressions guidance also documents property dereference and index access syntax for expression contexts. See Template expressions in Azure Pipelines and Azure Pipelines expressions.

QUESTION NO: 87

You need to use an Azure Pipelines pipeline to build and test an app and test the database of the app. The solution must meet the following requirements.

• Thetest stages must be run in parallel.

• The Publish_Test_Results stage must always be run.

• The test stages must be run after successful completion of the build stage.

• The Publish_Test_Results stage must be run after completion of all the test stages

Solution: You include the following elements in the YAML definition of the pipeline.



Does this meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct because the proposed stage dependency and execution configuration does not satisfy every required ordering and execution guarantee. Azure Pipelines uses the `dependsOn` relationship to form the stage dependency graph. To run the application and database tests in parallel only after a successful build, each test stage must independently depend on the build stage. The publishing stage must then explicitly depend on both test stages, ensuring that it is not scheduled until both branches of testing have completed. In addition, a publishing stage that must execute regardless of the result of preceding stages requires the `always()` condition. This condition is evaluated even when an upstream dependency fails, is canceled, or is skipped. A dependency arrangement that does not correctly express both test-stage dependencies and the final fan-in to publishing cannot guarantee the required behavior. Azure Pipelines stage dependencies and conditions are documented in Stages in Azure Pipelines and Specify conditions.

QUESTION NO: 88

You have a build pipeline in Azure Pipelines that uses different jobs to compile an application for 10 different architectures.

The build pipeline takes approximately one day to complete.

You need to reduce the time it takes to execute the build pipeline

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point

A. Move to a blue/green deployment pattern.

B. Create an agent pool.

C. Create a deployment group.

D. Reduce the size of the repository.

E. Increase the number of parallel jobs.

ANSWER: B E

Explanation:

Create an agent pool. is correct because an Azure Pipelines agent executes only one job at a time. A pool provides the logical grouping through which multiple agents can be made available to run the independent architecture-specific build jobs. With sufficient compatible agents registered in the pool, Azure Pipelines can assign separate compilation jobs to different agents concurrently rather than serializing them on one available agent. This directly reduces the elapsed duration of a pipeline whose work has already been divided into independent jobs.

Increase the number of parallel jobs. is also required because parallel-job capacity controls how many pipeline jobs the Azure DevOps organization is permitted to run simultaneously. Adding agents alone does not guarantee concurrent execution if the organization lacks sufficient parallel-job entitlement; surplus jobs remain queued until capacity is available. Therefore, combining a pool containing enough agents with adequate parallel-job capacity enables the ten architecture builds to execute in parallel, subject to dependencies and agent capabilities. Microsoft documents the one-job-per-agent behavior and agent-pool model in Azure Pipelines agents, and describes parallel-job capacity in Configure and pay for parallel jobs.

QUESTION NO: 89

Your company has an on-premises Bitbucket Server that is used for Git-based source control. The server is protected by a firewall that blocks inbound Internet traffic.

You plan to use Azure DevOps to manage the build and release processes

Which two components are required to integrate Azure DevOps and Bitbucket?

Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one port.

A. an External Git service connection

B. a Microsoft hosted agent

C. service hooks

D. a self- hosted agent

E. a deployment M group

ANSWER: A D

Explanation:

An External Git service connection is required so that Azure Pipelines has a securely stored endpoint definition and credentials for accessing the Git repository hosted outside Azure Repos. This connection enables the pipeline configuration to identify the on-premises Bitbucket Server repository and authenticate when Git operations such as cloning or fetching are performed. Azure DevOps service connections are designed to provide pipelines with authenticated access to external services without embedding credentials directly in pipeline YAML or scripts.

A self- hosted agent is required because the repository is on a private, on-premises network that cannot accept inbound connections from the Internet. The agent performs the checkout operation, so it must be installed on a machine with network connectivity to the Bitbucket Server. Self-hosted agents initiate outbound HTTPS communication to Azure DevOps to receive jobs, while retaining access to internal resources such as private Git servers. This architecture allows Azure DevOps to orchestrate builds and releases without exposing the Bitbucket Server through the firewall. Microsoft recommends self-hosted agents when pipeline jobs require access to resources that are unavailable to Microsoft-hosted agents. See Service connections in Azure Pipelines and Azure Pipelines agents.

QUESTION NO: 90

You have an app named App1 that will be deployed by using Azure Container Apps. You need to ensure that App1 can be deployed by using the blue-green deployment strategy. Which two actions should you perform in Container Apps? Each correct answer presents part of the solution. NOTE; Each correct selection is worth one point.

A. From Scale rule setting, set Max replicas to 2.

B. From Scale rule setting, set Min replicas to 2.

C. Set Revision Mode to Multiple

D. Configure continuous deployment.

E. Enable ingress.

ANSWER: C E

Explanation:

Set Revision Mode to Multiple is required because Azure Container Apps uses revisions to represent versioned snapshots of a container app. Blue-green deployment requires the currently serving revision (blue) and a newly deployed revision (green) to remain active at the same time. Multiple revision mode enables this concurrent-revision model and allows each revision to be independently activated, labeled, and managed while rollout decisions are made.

Enable ingress is also required because blue-green deployment depends on directing application traffic between the active revisions. When ingress is enabled, Azure Container Apps can route HTTP traffic and supports revision traffic splitting. This lets a team deploy the green revision without immediately sending it all production traffic, validate it, and then shift traffic from the blue revision to the green revision. Traffic can be moved in a controlled manner, including assigning all traffic to the new revision after validation or splitting traffic during a gradual transition. Together, multiple revision mode and ingress provide the revision coexistence and traffic-routing capabilities that implement blue-green deployments in Azure Container Apps.

For details, see Azure Container Apps revisions and traffic splitting in Azure Container Apps.

QUESTION NO: 91

Your company builds a multi tier web application.

>Youuse Azure DevOps and host the production application on Azure virtual machines.

Your team prepares an Azure Resource Manager template of the virtual machine that you mil use to test new features.

You need to create a staging environment in Azure that meetsthe following requirements:

• Minimizes the cost of Azure hosting

• Provisions the virtual machines automatically

• Use* the custom Azure Resource Manager template to provision the virtual machines

What should you do?

A. In Azure DevOps, configure new tasks in the release pipeline to create and delete the virtual machines in Azure DevTest Labs.

B. From Azure Cloud Shell, run Azure PowerShell commands to create and delete the new virtual machines in a staging resource group.

C. In Azure DevOps, configurenew tasks in the release pipeline to deploy to Azure Cloud Services.

D. In Azure Cloud Shell, run Azure CLI commands to create and delete the new virtual machines in a staging resource group.

ANSWER: A

Explanation:

In Azure DevOps, configure new tasks in the release pipeline to create and delete the virtual machines in Azure DevTest Labs. Azure DevTest Labs is designed for development and test workloads that need to be created on demand and removed when they are no longer needed. This lifecycle is well suited to a staging environment used to validate new application

features: the release pipeline can provision the required resources for the test, run the deployment and validation steps, and then delete the resources. Deleting the lab virtual machines after testing minimizes ongoing compute charges, while Azure DevTest Labs policies and quotas can further control consumption.

DevTest Labs supports reusable environments defined by Azure Resource Manager templates. A custom template can describe the virtual machine and its related Azure resources consistently, allowing each staging deployment to use the same approved configuration rather than relying on manual VM setup. Azure Pipelines can integrate this provisioning and cleanup into automated release stages, providing repeatable, short-lived test infrastructure. Microsoft describes using ARM templates to create DevTest Labs environments and provides Azure DevOps integration tasks for automated lab VM lifecycle operations. See Create DevTest Labs environments with ARM templates and Integrate Azure DevTest Labs with Azure Pipelines.

QUESTION NO: 92

You need to implement Project4.

What should you do first?

A. Add the FROM instruction in the Dockerfile file.

B. Add a Copy and Publish Build Artifacts task to the build pipeline.

C. Add a Docker task to the build pipeline.

D. Add the MAINTAINER instruction in the Dockerfile file.

ANSWER: C

Explanation:

Add a Docker task to the build pipeline is the appropriate first action because the pipeline must automate the container-image workflow required for Project4. Azure Pipelines provides the Docker@2 task specifically for Docker operations in a build or YAML pipeline. It can build an image from the repository's Dockerfile, apply the required tags, authenticate through an Azure Container Registry service connection, and push the resulting image to the registry. Using the `buildAndPush` command centralizes these steps in a repeatable CI process, so each qualifying build can produce a versioned image that is available for later deployment stages. The task should be configured with the Azure Container Registry service connection, image repository name, Dockerfile location, and tag strategy appropriate to the project. This establishes the essential integration between the application source, Docker build definition, and container registry before downstream release or deployment configuration is added. Microsoft documents that Docker@2 supports the `build`, `push`, and `buildAndPush` commands for container image automation. See the Docker@2 task reference and Microsoft's guidance on building and pushing Docker images to Azure Container Registry.

QUESTION NO: 93

Note: This question is part of a series of questions that present the same scenario. Each question in the series contains a unique solution that might meet the stated goals. Some question sets might have more than one correct solution, while others might not have a correct solution.

After you answer a question in this section, you will NOT be able to return to it. As a result, these questions will not appear in the review screen.

You integrate a cloud-hosted Jenkins server and a new Azure DevOps deployment.

You need Azure DevOps to send a notification to Jenkins when a developer commits changes to a branch in Azure Repos.

Solution: You create a service hook subscription that uses the build completed event.

Does this meet the goal?

A. Yes

B. No

ANSWER: B

Explanation:

No is correct. The required notification must be sent when changes are committed and pushed to a branch in an Azure Repos Git repository. For that purpose, Azure DevOps service hooks provide the *Code pushed* event. This event is raised when commits are pushed to a Git repository and can be filtered to a specific repository or branch, allowing Jenkins to be notified at the point the relevant code change is received by Azure Repos.

The proposed subscription instead uses *Build completed*, which is an event for the completion of an Azure DevOps build pipeline. It does not represent a repository push or commit event, and it is not guaranteed to occur when developers push code: a build might not be configured, might not be triggered by that branch, or might complete later than the push. Therefore, it cannot satisfy the requirement for a commit-based notification. Azure DevOps supports Jenkins as a service-hook consumer, where an appropriate event such as *Code pushed* can trigger a Jenkins job. Microsoft documents the available service-hook events, including Git push events, at Azure DevOps service hook events, and provides Jenkins integration guidance at Jenkins service hooks.

QUESTION NO: 94

You are developing an iOS application by using Azure DevOps.

You need to test the application manually on 10 devices without releasing the application to the public.

Which two actions should you perform? Each correct answer presents part of the solution.

NOTE: Each correct selection is worth one point.

A. Create a Microsoft Intune device compliance policy.

B. Deploy a certificate from an internal certification authority (CA) to each device.

C. Register the application in the iTunes store.

D. Onboard the devices into Microsoft Intune.

E. Distribute a new release of the application.

F. Register the IDs of the devices in the Apple Developer portal.

ANSWER: E F

Explanation:

Register the IDs of the devices in the Apple Developer portal. is required for an ad hoc iOS distribution workflow. Apple requires the unique device identifiers (UDIDs) of intended test devices to be registered to the developer account. Those devices can then be selected when creating the ad hoc provisioning profile used to sign the IPA. The resulting signed build is authorized to install and run on the specifically registered devices, allowing the team to test on the selected 10 devices without publishing the app in the public App Store. Apple documents device registration and the use of registered devices in provisioning resources at Register devices and Create an ad hoc provisioning profile.

Distribute a new release of the application. completes the private testing process after the build has been signed with the appropriate distribution certificate and provisioning profile. An Azure DevOps pipeline can produce the signed IPA and publish it through a controlled release or distribution mechanism so that the authorized testers can obtain the new build. This makes the application available for manual validation while retaining control over who receives it and without requiring a public-store release.