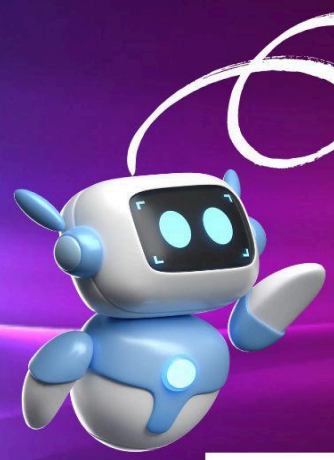
AZURE AI DEVOPS

ERRORS - TROUBLESHOOTING

PART 1



Azure DevOps

OpenAI

careerbytecode.substack.com
+32 - 471408908

1. Azure DevOps Pipeline Fails at Model Deployment Stage

Problem description:
During AI model deployment using Azure DevOps, the pipeline fails at the deployment stage.

Error:
```
Deployment failed: Model endpoint could not be reached
```

What we need to analyze:

- Network connectivity between Azure DevOps and Azure Machine Learning workspace.
- Logs from the Azure DevOps pipeline.
- Model deployment status in Azure ML.

How to troubleshoot:

1. Check if the ML workspace and Azure DevOps service connections are properly configured.
2. Verify network access and firewall rules.
3. Review logs for errors related to endpoint access.

How to resolve the issue:

- Reconfigure network settings to allow access.
- Ensure the correct authentication credentials are used.
- Restart the model deployment process.

Lessons learnt:

- Always validate service connections before triggering deployments.
- Configure proper network access for seamless communication.

---

2. Azure DevOps Pipeline Fails Due to Insufficient Compute Resources

Problem description:
AI model training pipeline fails because of resource exhaustion.

Error:

```
Error: Insufficient compute resources available
```

What we need to analyze:

- Available compute resources in Azure ML workspace.
- Quotas for CPU, GPU, and memory.
- Any other jobs running in the workspace.

How to troubleshoot:

1. Check Azure ML resource utilization.
2. Review Azure subscription quotas.
3. Identify running jobs that might be consuming resources.

How to resolve the issue:

- Increase compute instance size or use a different virtual machine type.
- Request quota increase from Azure support.
- Schedule training jobs at non-peak times.

Lessons learnt:

- Monitor compute resource availability before running jobs.
- Plan resource allocation efficiently.

3. Model Registry Access Denied in Azure DevOps

Problem description:
Azure DevOps pipeline cannot access the Azure Machine Learning model registry.

Error:

```
Access denied: Unable to retrieve model from registry
```

What we need to analyze:

- Service principal permissions.
- Access control list (ACL) of the model registry.
- Azure DevOps service connection authentication.

How to troubleshoot:

1. Verify if the service principal has Contributor or Reader access to the ML workspace.
2. Check the model registry ACL settings.
3. Inspect Azure DevOps service connections for authentication failures.

How to resolve the issue:

- Grant the necessary permissions to the service principal.
- Update the Azure DevOps service connection credentials.
- Re-authenticate and retry the operation.

Lessons learnt:

- Ensure proper access permissions are set before deployment.
- Regularly audit and validate service connections.

---

4. Failure in Azure DevOps ML Model Training Due to Incorrect Environment

Problem description:
Pipeline fails because the wrong environment is used for AI model training.

Error:
```
Error: Dependency mismatch detected
```

What we need to analyze:

- The Python environment and dependencies used in Azure ML.
- The environment configuration file (.yml or requirements.txt).
- Pipeline logs for dependency-related errors.

How to troubleshoot:

1. Compare the environment configuration file with the actual environment.
2. Check for missing or incompatible dependencies.
3. Ensure that the Azure ML compute instance has the correct environment.

How to resolve the issue:

- Update the environment configuration file with the correct dependencies.
- Rebuild and validate the environment in Azure ML.
- Test the training job with a validated environment.

Lessons learnt:

- Always validate environment dependencies before training models.
- Maintain version control for environment configurations.

---

5. Data Ingestion Failure in Azure DevOps AI Pipeline

Problem description:
Data ingestion step fails due to incorrect data source or permission issues.

Error:

```
Error: Unable to access the specified data source
```

What we need to analyze:

- The data source URL and access credentials.
- Service principal permissions.
- Azure storage firewall rules.

How to troubleshoot:

1. Verify data source accessibility manually.
2. Check if the service principal has the required permissions.
3. Review storage logs for access-related errors.

How to resolve the issue:

- Grant necessary permissions to the service principal.
- Update firewall rules to allow access.
- Validate the data source and retry ingestion.

Lessons learnt:

- Always test data access permissions before running pipelines.
- Use managed identities for secure authentication.

6. Azure DevOps Model Deployment Fails Due to Invalid Model Format

Problem description:
When deploying an AI model, the pipeline fails due to an invalid model format.

Error:

```
Error: Unsupported model format detected
```

What we need to analyze:

- The model file format and compatibility with Azure ML.
- The model export process in the training pipeline.
- The inference environment configuration.

How to troubleshoot:

1. Check if the model is exported in a compatible format (ONNX, TensorFlow, PyTorch, etc.).
2. Verify the model packaging process in the pipeline.
3. Compare the model's framework with the target deployment environment.

How to resolve the issue:

- Convert the model to a supported format before deployment.
- Update the training script to export the correct model format.
- Ensure the inference environment has the required dependencies.

Lessons learnt:

- Always validate model formats before deployment.
- Maintain consistency between training and inference environments.

---

7. Azure DevOps Pipeline Fails Due to Expired Service Principal Credentials

Problem description:
Pipeline execution fails because the service principal credentials have expired.

Error:

```
Error: Authentication failed due to expired credentials
```

What we need to analyze:

- The expiration date of the service principal's secret or certificate.
- The Azure DevOps service connection settings.
- Azure AD logs for authentication failures.

How to troubleshoot:

1. Check the service principal's expiration date in Azure AD.
2. Validate the Azure DevOps service connection authentication method.
3. Inspect logs for expired token errors.

How to resolve the issue:

- Renew the service principal's credentials in Azure AD.
- Update the Azure DevOps service connection with the new credentials.
- Use managed identities to reduce credential management overhead.

Lessons learnt:

- Set up automated credential rotation to prevent expiration issues.
- Use managed identities for authentication where possible.

---

8. Azure DevOps Fails to Retrieve Dataset from Azure Data Lake

Problem description:
The pipeline cannot access data stored in Azure Data Lake due to permission issues.

Error:

```
Error: Access denied while retrieving data from Azure Data Lake
```

What we need to analyze:

- Service principal or managed identity permissions.
- Azure Data Lake firewall rules.
- Data Lake storage access policies.

How to troubleshoot:

1. Check if the service principal has at least Reader access to the Data Lake.
2. Review Azure Data Lake access control lists (ACLs).
3. Verify firewall and private endpoint settings.

How to resolve the issue:

- Grant the required access permissions to the service principal.
- Adjust the ACLs to allow read access.
- Configure firewall rules to allow pipeline access.

Lessons learnt:

- Always validate data access before executing pipelines.
- Implement role-based access control (RBAC) for secure access management.

---

9. Azure DevOps Model Performance Degradation After Deployment

Problem description:
After deployment, the AI model shows degraded performance compared to training results.

Error:

```
Warning: Model performance significantly lower than expected
```

What we need to analyze:

- Differences between training and inference datasets.
- Model versioning history.
- Compute resource variations affecting inference speed.

How to troubleshoot:

1. Compare training data with real-world inference data.
2. Check if the correct model version is deployed.
3. Monitor CPU/GPU utilization to detect resource bottlenecks.

How to resolve the issue:

- Retrain the model with more diverse data.
- Optimize inference compute resources.
- Use model monitoring tools to track drift and retrain when necessary.

Lessons learnt:

- Regularly monitor model drift to maintain accuracy.
- Use A/B testing before full-scale deployment.

10. Azure DevOps Model Training Fails Due to Incorrect Compute Target

Problem description:
Model training fails because the assigned compute target is unavailable or misconfigured.

Error:

```
Error: Compute target not found or not available
```

What we need to analyze:

- The availability of the specified compute instance in Azure ML.
- The region of the compute target.
- The compute instance state (running, stopped, deleted).

How to troubleshoot:

1. Check if the compute instance exists in Azure ML.
2. Verify that the compute target is running and accessible.
3. Review Azure resource quotas for any exceeded limits.

How to resolve the issue:

- Start the compute instance if it is stopped.
- Select an alternative compute target if needed.
- Increase quota limits if the resources are exhausted.

Lessons learnt:

- Always check compute resource availability before launching training jobs.
- Use auto-scaling to manage compute requirements dynamically.

---

11. Azure DevOps Pipeline Fails Due to Incompatible Python Version

Problem description:
The AI pipeline fails during execution due to an incompatible Python version in the Azure ML environment.

Error:

```
Error: Unsupported Python version detected
```

What we need to analyze:

- The Python version specified in the Azure ML environment.
- The version used in the training and inference scripts.
- The compatibility of libraries with the chosen Python version.

How to troubleshoot:

1. Check the Python version in the Azure ML environment.
2. Compare it with the Python version used during development.
3. Review dependency compatibility issues.

How to resolve the issue:

- Update the Azure ML environment to use the correct Python version.
- Modify the training and inference scripts to be compatible with the available version.
- Rebuild the environment if necessary.

Lessons learnt:

- Maintain consistency between development and production environments.
- Specify the required Python version in the environment configuration file.

---

12. Azure DevOps Model Training Fails Due to Outdated Azure SDK Version

Problem description:
The pipeline fails because the Azure SDK version used is outdated and no longer supported.

Error:

```
Error: Unsupported Azure SDK version detected
```

What we need to analyze:

- The Azure SDK version installed in the environment.
- The compatibility of the SDK version with the Azure ML service.
- Any deprecated features causing failures.

How to troubleshoot:

1. Identify the Azure SDK version being used.
2. Check the Azure documentation for compatibility updates.
3. Look for API changes in the logs.

How to resolve the issue:

- Upgrade the Azure SDK to a supported version.
- Modify scripts to use updated API methods.
- Test the pipeline with the new SDK version.

Lessons learnt:

- Regularly update SDK versions to prevent compatibility issues.
- Use virtual environments to manage dependencies effectively.

---

13. Azure DevOps Model Deployment Fails Due to Missing Dependencies

Problem description:
Deployment fails because the necessary dependencies are not installed in the Azure ML environment.

Error:

```
Error: Missing module <dependency_name>
```

What we need to analyze:

- The dependency list in requirements.txt or conda.yaml.
- The installed packages in the Azure ML environment.
- The logs for missing or incorrect package versions.

How to troubleshoot:

1. Compare the required dependencies with the installed ones.
2. Check for package version mismatches.
3. Verify if dependencies are installed in the correct virtual environment.

How to resolve the issue:

- Add missing dependencies to requirements.txt or conda.yaml.
- Rebuild the environment with the updated dependency list.
- Use a custom Docker container with all required dependencies pre-installed.

Lessons learnt:

- Keep a well-maintained dependency file to avoid missing modules.
- Validate environment dependencies before deployment.

---

14. Azure DevOps Pipeline Fails Due to Incorrect Model Path

Problem description:
The pipeline fails because the model file path specified in the script is incorrect.

Error:

```
Error: Model file not found at specified path
```

What we need to analyze:

- The model storage location in Azure ML.
- The path used in the deployment script.
- The workspace and file system structure.

How to troubleshoot:

1. Verify if the model exists in the specified location.
2. Check the path formatting in the script.
3. Ensure that the model is properly registered and available.

How to resolve the issue:

- Update the script with the correct model path.
- Re-upload the model if it was accidentally deleted.
- Use the Azure ML SDK to dynamically fetch the model path.

Lessons learnt:

- Always verify file paths before running pipelines.
- Use dynamic model retrieval methods to avoid hardcoded paths.

---

15. Azure DevOps Pipeline Fails Due to Azure ML Workspace Misconfiguration

Problem description:
The AI pipeline fails because the Azure ML workspace is not correctly configured.

Error:

```
Error: Azure ML workspace not found
```

What we need to analyze:

- The workspace name, resource group, and subscription details.
- The authentication method used.
- The Azure DevOps service connection settings.

How to troubleshoot:

1. Check if the workspace exists in the Azure portal.
2. Verify that the correct resource group and subscription are being used.
3. Ensure that the service principal has access to the workspace.

How to resolve the issue:

- Update the workspace details in the pipeline configuration.
- Grant the necessary permissions to the service principal.
- Reauthenticate the Azure DevOps service connection.

Lessons learnt:

- Always validate workspace configurations before pipeline execution.
- Store workspace details in environment variables to avoid misconfigurations.

---

16. Azure DevOps Model Scoring Endpoint Returns Incorrect Predictions

Problem description:
The deployed model returns incorrect predictions compared to expected results.

Error:

```
Warning: Model output does not match expected values
```

What we need to analyze:

- The input data format used in the inference request.
- The preprocessing steps applied before inference.
- The model version deployed.

How to troubleshoot:

1. Compare the inference data with the training data format.
2. Check if preprocessing steps are applied correctly.
3. Verify that the correct model version is deployed.

How to resolve the issue:

- Update the inference script to match the training data preprocessing.
- Deploy the correct model version.
- Conduct thorough model validation before deployment.

Lessons learnt:

- Ensure consistency in data preprocessing during training and inference.
- Use automated tests to validate model outputs before deployment.

---

17. Azure DevOps Model Retraining Fails Due to Data Schema Mismatch

Problem description:
The model retraining pipeline fails because the new dataset schema does not match the original.

Error:

```
Error: Data schema mismatch detected
```

What we need to analyze:

- The schema of the new dataset compared to the original.
- Feature engineering steps applied in the pipeline.
- Data validation logs.

How to troubleshoot:

1. Compare column names and data types between old and new datasets.
2. Check if any preprocessing steps modify the schema.
3. Review error logs for missing or extra columns.

How to resolve the issue:

- Update the preprocessing script to handle schema changes.
- Modify the model input pipeline to accept new data formats.
- Implement schema validation checks before training.

Lessons learnt:

- Always validate dataset schemas before retraining models.
- Implement automated schema checks in the data pipeline.

---

18. Azure DevOps Pipeline Fails Due to Insufficient Compute Resources

Problem description:
The AI model training pipeline fails due to a lack of available compute resources in the Azure ML environment.

Error:

```
Error: Insufficient compute resources available
```

What we need to analyze:

- The compute cluster type and its assigned resources.
- The current workload and availability of compute nodes.
- The Azure ML resource quota limits for the subscription.

How to troubleshoot:

1. Check the compute cluster's status in the Azure portal.
2. Identify if the cluster has reached its max node limit.
3. Verify if the quota limits have been exceeded for the subscription.

How to resolve the issue:

- Increase the node count in the compute cluster.
- Upgrade to a higher SKU for better processing power.
- Request a quota increase from Azure Support if needed.

Lessons learnt:

- Regularly monitor compute usage to avoid unexpected failures.
- Set up auto-scaling for efficient resource allocation.

---

19. Azure DevOps Pipeline Execution Fails Due to Authentication Issues

Problem description:
The pipeline fails due to authentication failures when accessing Azure ML services.

Error:

```
Error: Unauthorized access to Azure ML workspace
```

What we need to analyze:

- The service principal permissions assigned in Azure AD.
- The Azure DevOps service connection settings.
- The authentication method used in the pipeline.

How to troubleshoot:

1. Check if the service principal has the correct role assignments.
2. Verify the credentials used in the pipeline.
3. Look for authentication errors in the logs.

How to resolve the issue:

- Update role-based access control (RBAC) settings.
- Reconfigure the Azure DevOps service connection.
- Use managed identities for more secure authentication.

Lessons learnt:

- Always validate authentication settings before executing pipelines.
- Use role-based access control to ensure secure access management.

---

20. Azure DevOps Pipeline Stalls Due to Long Running Jobs

Problem description:
The pipeline takes much longer than expected, stalling at certain stages.

Error:

```
Warning: Pipeline execution taking too long
```

What we need to analyze:

- The execution time of each pipeline step.
- The efficiency of the model training process.
- The performance of data preprocessing steps.

How to troubleshoot:

1. Analyze the logs to identify bottlenecks.
2. Check if large datasets are causing delays.
3. Review the compute resource utilization.

How to resolve the issue:

- Optimize data preprocessing by reducing redundant transformations.
- Use distributed computing techniques for large workloads.
- Tune hyperparameters to speed up model training.

Lessons learnt:

- Optimize pipeline efficiency to avoid resource wastage.
- Use logging and monitoring to identify slow-performing tasks.

---

21. Azure Machine Learning Model Deployment Fails in DevOps Pipeline

Problem description:
The AI model fails to deploy to an Azure ML endpoint from the DevOps pipeline.

Error:

```
Error: Deployment failed. No available compute resources
```

What we need to analyze:

- The target deployment compute instance or AKS cluster.
- The model's resource requirements (CPU, GPU, memory).
- The containerization and dependencies of the model.

How to troubleshoot:

1. Check if the deployment target (VM, AKS, or ACI) is running.
2. Review the logs for dependency installation failures.
3. Verify if resource allocation is sufficient for model execution.

How to resolve the issue:

- Allocate more resources to the deployment target.
- Ensure dependencies are correctly packaged in the model container.
- Use auto-scaling AKS clusters for better resource management.

Lessons learnt:

- Always validate deployment targets before pipeline execution.
- Use containerization best practices to minimize dependency issues.

---

22. Azure AI Model Training Pipeline Fails Due to Data Access Issues

Problem description:
The pipeline fails when trying to access training data from Azure Blob Storage.

Error:

```
Error: Permission denied when accessing Azure Blob Storage
```

What we need to analyze:

- The storage account access policies.
- The service principal or managed identity permissions.
- The network security settings (firewalls, VNET restrictions).

How to troubleshoot:

1. Check if the service principal has Storage Blob Data Reader/Contributor role.
2. Ensure the pipeline is using the correct authentication method.
3. Verify that firewall and network rules allow access.

How to resolve the issue:

- Assign the necessary RBAC roles to the service principal.
- Use managed identities for more secure authentication.
- Modify firewall rules to allow access from trusted networks.

Lessons learnt:

- Properly configure access control before running AI pipelines.
- Use managed identities instead of storing credentials in scripts.

---

23. Azure DevOps Build Pipeline Fails Due to Python Dependency Conflicts

Problem description:
The DevOps pipeline fails during the build process due to conflicting Python package versions.

Error:
`Error: Incompatible package versions detected in environment.yaml`

What we need to analyze:

- The dependencies listed in the `requirements.txt` or `environment.yaml`.
- The compatibility of libraries used in the AI model.
- The logs to find the exact package causing the conflict.

How to troubleshoot:

1. Run `pip check` or `conda list --explicit` locally to check conflicts.
2. Review the logs for the specific package causing the issue.
3. Test installing dependencies in an isolated virtual environment.

How to resolve the issue:

- Pin package versions to avoid unexpected upgrades.
- Use `pip-compile` or `conda-lock` to generate a stable dependency list.
- Use a Docker image with pre-installed dependencies for consistency.

Lessons learnt:

- Always test package installations in a virtual environment before deployment.
- Lock dependency versions to avoid unexpected failures.

---

24. Azure DevOps Pipeline Fails Due to Incorrect Environment Variables

Problem description:
The DevOps pipeline fails to execute correctly because required environment variables are not set or are incorrect.

Error:

```
Error: Environment variable <VARIABLE_NAME> not found or incorrect value provided.
```

What we need to analyze:

- The environment variables configured in the pipeline.
- The key-value pairs stored in Azure Key Vault or pipeline variables.
- The scope of variables (global vs. local) in the pipeline YAML file.

How to troubleshoot:

1. Check if the environment variables are defined in the pipeline settings.
2. Validate if sensitive variables are stored securely in Azure Key Vault.
3. Ensure that the correct syntax is used in the YAML pipeline file (`variables:` section).

How to resolve the issue:

- Define missing environment variables in the Azure DevOps pipeline.
- Use Azure Key Vault for secure storage of sensitive information.
- Use the correct syntax and scope when referencing variables in the pipeline.

Lessons learnt:

- Always verify that necessary environment variables are properly set before pipeline execution.
- Use secure storage mechanisms like Azure Key Vault to manage sensitive credentials.

---

25. Azure DevOps Pipeline Fails Due to Incompatible Azure CLI Version

Problem description:
The pipeline fails because the Azure CLI version in the agent is outdated or incompatible with the commands being used.

Error:
```
Error: Command 'az ml model register' is not recognized in the current CLI version.
```

What we need to analyze:

- The Azure CLI version installed on the pipeline agent.
- The compatibility of CLI commands used in the pipeline.
- The available CLI versions and their support for specific commands.

How to troubleshoot:

1. Check the Azure CLI version by running `az --version`.
2. Compare the CLI version with the Azure ML SDK documentation.
3. Test the same command locally with an updated CLI version.

How to resolve the issue:

- Upgrade the Azure CLI version using `az upgrade`.
- Ensure that the pipeline agent has the latest CLI version installed.
- Use a Docker image with a predefined Azure CLI version to avoid inconsistencies.

Lessons learnt:

- Always validate CLI versions before executing Azure DevOps pipelines.
- Use containerized environments to ensure consistent CLI behavior across builds.

---

26. Azure DevOps Release Pipeline Fails Due to Incorrect Service Connection Permissions

Problem description:
The release pipeline fails when attempting to deploy AI models due to incorrect permissions in the Azure DevOps service connection.

Error:

```
Error: Failed to authenticate service connection. Insufficient permissions.
```

What we need to analyze:

- The role assigned to the service principal in Azure AD.
- The permissions granted in the Azure DevOps service connection.
- The logs for authentication failure details.

How to troubleshoot:

1. Check the service principal's role assignments in Azure AD.
2. Ensure that the service connection has the required permissions for deployment.
3. Verify that the correct authentication method is being used.

How to resolve the issue:

- Grant the necessary RBAC roles to the service principal (e.g., Contributor or Owner).
- Reconfigure the service connection with correct authentication credentials.
- Use managed identities for better security and automated authentication.

Lessons learnt:

- Always validate service principal permissions before executing deployments.
- Use managed identities to minimize credential management overhead.

---

27. Azure DevOps Pipeline Fails Due to GPU Unavailability in Compute Cluster

Problem description:
The pipeline fails during AI model training because the required GPU resources are unavailable in the Azure Machine Learning compute cluster.

Error:
```
Error: No GPU instances available. Unable to allocate requested resources.
```

What we need to analyze:

- The type and availability of GPU-enabled VMs in the compute cluster.
- The quotas assigned for GPU resources in the Azure subscription.
- The pipeline configuration specifying GPU requirements.

How to troubleshoot:

1. Check the compute cluster's node status in Azure Machine Learning Studio.
2. Verify the Azure quota for GPU instances using `az vm list-usage`.
3. Confirm that the correct VM SKU (e.g., `Standard_NC6`) is specified in the deployment YAML.

How to resolve the issue:

- Increase the GPU quota limit in Azure by requesting additional resources.
- Use an alternative region where GPU resources are more readily available.
- Consider fallback options, such as CPU-based training, if GPU is unavailable.

Lessons learnt:

- Always check GPU availability before executing a pipeline that requires acceleration.
- Request quota increases in advance for high-demand GPU resources.

28. Azure AI Model Deployment Fails Due to Container Image Build Failure

Problem description:
The model deployment pipeline fails because the container image for the model fails to build properly.

Error:
`Error: Docker build failed due to missing dependencies.`

What we need to analyze:

- The `Dockerfile` used to build the container.
- The dependencies listed in the `requirements.txt` or `conda.yaml`.
- The logs from the container registry build process.

How to troubleshoot:

1. Run the Docker build command locally to replicate the issue.
2. Check for missing dependencies or incorrect package versions.
3. Ensure that the base image specified in the `Dockerfile` is available and supported.

How to resolve the issue:

- Update the `Dockerfile` to include all necessary dependencies.
- Use a specific version of the base image to avoid compatibility issues.
- Store the container image in Azure Container Registry for consistent deployment.

Lessons learnt:

- Always test container builds locally before running them in a DevOps pipeline.
- Use versioned base images to avoid unexpected changes in dependencies.

29. Azure Machine Learning Model Inference is Slow Due to Inefficient Code

Problem description:
The deployed AI model takes too long to return predictions when hosted as an Azure ML web service.

Error:
No specific error, but inference latency is significantly high.

What we need to analyze:

- The efficiency of the model inference code.
- The hardware used for serving the model (CPU vs. GPU).
- The batch size and optimization techniques applied.

How to troubleshoot:

1. Profile the inference code using tools like `cProfile` in Python.
2. Test inference performance with different batch sizes.
3. Compare execution times on different VM SKUs (CPU vs. GPU).

How to resolve the issue:

- Optimize the model by converting it to ONNX format.
- Use GPU-enabled compute instances for faster inference.
- Enable caching mechanisms to reduce redundant computations.

Lessons learnt:

- Model optimization techniques can significantly improve inference speed.
- Selecting the right compute resources is crucial for performance.

---

30. Azure DevOps Pipeline Fails Due to Insufficient Permissions to Access Azure Key Vault

Problem description:
The DevOps pipeline fails when trying to retrieve secrets from Azure Key Vault due to inadequate permissions.

Error:
`Error: Operation failed due to insufficient privileges. Access denied to Azure Key Vault.`

What we need to analyze:

- The permissions assigned to the Azure DevOps service principal or managed identity.
- The Key Vault access policies and RBAC roles.
- The pipeline logs to confirm the authentication failure details.

How to troubleshoot:

1. Check if the service principal has the `Get` and `List` permissions on Key Vault secrets.
2. Verify the assigned RBAC role using `az role assignment list`.
3. Test access manually using the Azure CLI:

```
Unset
az keyvault secret show --name <secret-name> --vault-name <vault-name>
```

How to resolve the issue:

- Assign the `Key Vault Secrets User` or `Key Vault Administrator` role to the service principal.
- If using a managed identity, enable access in the Key Vault access policy.
- Ensure that the correct authentication method (Managed Identity or Service Principal) is used in the pipeline.

Lessons learnt:

- Always verify Key Vault permissions before pipeline execution.
- Use managed identities instead of service principals to simplify authentication.

---

31. Azure DevOps Build Pipeline Fails Due to Timeout in Artifact Upload

Problem description:
The build pipeline fails because the upload of build artifacts exceeds the timeout limit.

Error:

```
Error: Timeout while uploading artifact to Azure DevOps.
```

What we need to analyze:

- The artifact size and network bandwidth.
- The pipeline agent's available storage and connectivity status.
- The Azure DevOps artifact retention settings.

How to troubleshoot:

1. Check the size of the artifact being uploaded.
2. Verify the network speed and connectivity between the agent and Azure DevOps.
3. Increase the artifact upload timeout settings in the pipeline YAML file.

How to resolve the issue:

- Reduce the size of the artifact by compressing files before uploading.
- Increase the timeout settings in the pipeline.
- Use an alternative storage solution, such as Azure Blob Storage, for large artifacts.

Lessons learnt:

- Optimizing artifact size improves pipeline efficiency.
- Monitor network speed and storage capacity on pipeline agents.

---

32. Azure ML Model Deployment Fails Due to Unsupported Framework Version

Problem description:
The deployment of an AI model fails because the framework version used in the pipeline does not match the supported version in Azure ML.

Error:

```
Error: Framework version <x.y.z> is not supported in Azure ML Compute.
```

What we need to analyze:

- The framework version specified in the model environment.
- The Azure ML environment configuration for compatibility.
- The available versions supported in the Azure ML workspace.

How to troubleshoot:

1. Check the framework version used in the model with:

```
Unset
pip show tensorflow  # Example for TensorFlow
```

2. Compare the version with the supported versions listed in Azure ML documentation.
3. Test the model deployment locally using the same framework version.

How to resolve the issue:

- Update the Azure ML environment to match the correct framework version.
- Use a compatible Docker image with the required dependencies.
- If necessary, downgrade or upgrade the framework version in the training environment.

Lessons learnt:

- Always ensure model framework compatibility before deployment.
- Use Azure ML curated environments to avoid version mismatches.

---

33. Azure DevOps Pipeline Fails Due to Incorrect Python Package Version in Virtual Environment

Problem description:
A machine learning pipeline in Azure DevOps fails due to an incompatible or missing Python package version inside the virtual environment.

Error:
`ModuleNotFoundError: No module named 'pandas'`
or
`ImportError: cannot import name 'SomeFunction' from 'tensorflow'`

What we need to analyze:

- The Python package versions in the pipeline execution environment.
- The `requirements.txt` or `conda.yaml` used for dependency installation.
- The logs from the `pip install` or `conda install` command in the pipeline.

How to troubleshoot:

1. Check the installed package versions in the Azure DevOps agent using:

```
Unset
pip list
```

2. Compare the versions with those specified in `requirements.txt`.
3. If using Conda, verify dependencies with:

```
Unset
conda list
```

How to resolve the issue:

- Explicitly define package versions in `requirements.txt` or `conda.yaml`.
- Use a virtual environment to isolate dependencies:

```
Unset
python -m venv myenv

source myenv/bin/activate  # (Linux/macOS)
```

```
myenv\Scripts\activate  # (Windows)
```

- Ensure that the pipeline uses the correct Python environment with the needed dependencies.

Lessons learnt:

- Always specify exact package versions to avoid compatibility issues.
- Use virtual environments to prevent dependency conflicts.

---

34. Azure AI Model Deployment Fails Due to Insufficient Memory on Compute Instance

Problem description:
The deployment of an AI model fails because the compute instance does not have enough memory to load the model.

Error:
`Error: OOM (Out of Memory) exception while loading the model.`

What we need to analyze:

- The size of the model file and memory requirements.
- The type of Azure compute instance used for inference.
- Logs for memory usage at the time of failure.

How to troubleshoot:

1. Check the model file size and compare it with the available RAM.
2. Monitor memory usage using:

```
Unset
free -m  # (Linux)

Get-WMIObject Win32_OperatingSystem | Select-Object
TotalVisibleMemorySize, FreePhysicalMemory  # (Windows)
```

3. Test model loading on a local machine with similar memory constraints.

How to resolve the issue:

- Optimize the model by converting it to ONNX format.
- Use a higher-memory Azure VM SKU (e.g., `Standard_D8s_v3`).
- Enable model quantization to reduce memory footprint.

Lessons learnt:

- Large models require careful memory planning.
- Use efficient model formats for deployment.

---

35. Azure DevOps Pipeline Fails Due to Expired Azure Service Principal Credentials

Problem description:
The pipeline fails because the service principal used for authentication has expired credentials.

Error:

```
Error: Authentication failed. Service principal credentials are invalid or expired.
```

What we need to analyze:

- The expiration date of the service principal credentials.
- Whether the correct service principal is being used in the pipeline.
- Azure Active Directory logs for authentication failures.

How to troubleshoot:

1. Verify the expiration date of the service principal using:

```
Unset
az ad sp credential list --id <service-principal-id>
```

2. Check if the service principal is assigned the correct role in Azure.
3. Inspect pipeline logs for authentication errors.

How to resolve the issue:

- Renew the service principal credentials and update them in Azure DevOps.
- Use Managed Identity instead of service principals where possible.
- Implement a proactive alert system for credential expiration.

Lessons learnt:

- Regularly monitor service principal expiration dates.
- Prefer Managed Identities to reduce authentication issues.

---

36. Azure Machine Learning Pipeline Fails Due to Storage Account Firewall Restrictions

Problem description:
An Azure ML pipeline fails when trying to access a storage account because firewall restrictions prevent access.

Error:

```
Error: Access to Azure Storage account '<storage-account-name>' is denied due to firewall restrictions.
```

What we need to analyze:

- The storage account firewall settings in the Azure portal.
- The Azure ML compute environment's network access settings.
- Whether private endpoints or VNET integration is enabled.

How to troubleshoot:

1. Check the storage account's firewall settings under *Networking* > *Firewalls and virtual networks*.
2. Verify whether the Azure ML workspace or compute instance is allowed access.
3. Use the Azure CLI to test access from the pipeline agent:

```
Unset
az storage blob list --account-name <storage-account-name>
--container-name <container-name>
```

How to resolve the issue:

- Allow the required IP ranges or virtual networks in the storage account firewall.
- Enable *Allow Azure services on the trusted services list* in the storage account settings.
- Use a private endpoint for more secure access.

Lessons learnt:

- Always validate network access settings before running ML pipelines.
- Use private endpoints for security while ensuring connectivity.

37. Azure DevOps Release Pipeline Fails Due to Locked Resource Group

Problem description:
A release pipeline in Azure DevOps fails when deploying to a resource group that is locked as *Read-only* or *Delete locked*.

Error:
`Error: Deployment failed due to resource group '<resource-group-name>' being locked.`

What we need to analyze:

- The lock settings on the resource group in the Azure portal.
- The role-based access control (RBAC) settings for the service principal.
- The deployment logs for more details on permission errors.

How to troubleshoot:

1. Check the resource group's lock settings under *Settings* > *Locks* in the Azure portal.
2. Identify who applied the lock and why.
3. Test if the pipeline service principal has write access using:

```
Unset
az role assignment list --assignee <service-principal-id>
```

How to resolve the issue:

- Remove the lock temporarily if appropriate.
- Update the pipeline to deploy to an unlocked resource group.
- If the lock is necessary, ensure proper access control policies are defined.

Lessons learnt:

- Review resource locks before executing automated deployments.
- Communicate with security teams before modifying locked resources.

38. Azure AI Model Fails to Load Due to Missing GPU Drivers in Compute Cluster

Problem description:
An AI model deployment fails because the compute cluster does not have the necessary GPU drivers installed.

Error:
`Error: CUDA driver not found. Please install the correct NVIDIA drivers for GPU usage.`

What we need to analyze:

- The GPU driver version installed on the compute instance.
- The container or virtual machine image being used for model inference.
- The Azure Machine Learning workspace configuration.

How to troubleshoot:

1. Check the installed GPU drivers using:

```
Unset
nvidia-smi
```

2. Ensure that the compute instance is a GPU-enabled SKU (e.g., `NC`, `ND` series).
3. Test GPU access in the model container:

```
Unset
python -c "import torch; print(torch.cuda.is_available())"
```

How to resolve the issue:

- Install the correct NVIDIA drivers:

```
Unset
sudo apt-get install -y nvidia-driver-470
```

- Use a pre-configured Azure ML GPU environment.
- Ensure the Docker container includes CUDA libraries if running in a containerized setup.

Lessons learnt:

- Always validate GPU availability before deploying ML workloads.
- Use Azure ML curated environments to avoid manual driver installations.

39. Azure DevOps Pipeline Fails Due to Overlapping Terraform State Files

Problem description:
Terraform deployment in Azure DevOps fails because multiple users or processes are trying to modify the same state file simultaneously.

Error:

```
Error: Terraform state file is locked. Another process is modifying the state.
```

What we need to analyze:

- The Terraform backend configuration (Azure Storage, Terraform Cloud, etc.).
- Whether multiple pipelines or developers are modifying the same state file.
- Terraform logs to identify the process holding the lock.

How to troubleshoot:

1. Check the state lock using:

```
Unset
terraform state list
```

2. Inspect the backend storage account to ensure it is properly configured for locking.
3. Review concurrent Terraform runs in Azure DevOps.

How to resolve the issue:

- Use a remote backend with state locking (e.g., Azure Storage with blob locks).
- Ensure only one pipeline modifies the state at a time.
- Manually unlock the state file if needed:

```
Unset
terraform force-unlock <lock-id>
```

Lessons learnt:

- Always configure remote state locking in Terraform.
- Avoid concurrent state modifications in CI/CD pipelines.

40. Azure DevOps Pipeline Fails Due to Expired Personal Access Token (PAT) in Git Repository

Problem description:
A pipeline fails when trying to clone or push to an Azure Repos or GitHub repository due to an expired Personal Access Token (PAT).

Error:

```
fatal: Authentication failed for
'https://dev.azure.com/organization/project/_git/repository'
```

What we need to analyze:

- The expiration date of the PAT used for authentication.
- Whether the correct PAT scope and permissions are assigned.
- The pipeline logs to check if the correct credentials are used.

How to troubleshoot:

1. Check the expiration of the PAT in Azure DevOps under *User Settings* > *Personal Access Tokens*.
2. Validate that the pipeline is using the latest PAT.
3. If using GitHub, check the token under *Settings* > *Developer settings* > *Personal Access Tokens*.

How to resolve the issue:

- Generate a new PAT with the required permissions and update it in the pipeline.
- Use Azure DevOps Service Connections instead of PATs for authentication.
- Store the PAT securely in Azure Key Vault or pipeline secrets.

Lessons learnt:

- Regularly update and monitor PAT expiration dates.
- Prefer service connections over PATs for better security.

---

41. Azure Machine Learning Model Deployment Fails Due to Missing Dependencies in Environment

Problem description:
An ML model deployment fails because required libraries are missing from the Azure ML environment.

Error:
`ModuleNotFoundError: No module named 'sklearn'`

What we need to analyze:

- The environment configuration file (`environment.yml`, `requirements.txt`).
- The logs from environment setup during deployment.
- The container image used for inference.

How to troubleshoot:

1. Check the installed libraries using:

```
Unset
pip list
```

2. Verify that the correct environment is being used in the Azure ML workspace.
3. If using a container, check the base image for missing dependencies.

How to resolve the issue:

- Add missing dependencies in `requirements.txt`:

```
Unset
scikit-learn==1.0.2
```

- Use a pre-configured Azure ML environment with all required packages.
- Test the environment locally before deploying.

Lessons learnt:

- Always specify dependencies explicitly in environment files.
- Test ML environments before deploying to production.

---

42. Azure AI Model Training Fails Due to Insufficient Disk Space on Compute Node

Problem description:
An AI model training job fails because the compute node runs out of disk space.

Error:

```
Error: No space left on device
```

What we need to analyze:

- The disk usage on the Azure compute node.
- The size of the dataset and model checkpoints.
- The Azure ML workspace settings.

How to troubleshoot:

1. Check disk usage using:

```
Unset
df -h
```

2. Verify the dataset size compared to available disk space.
3. Inspect log files that might be consuming excess storage.

How to resolve the issue:

- Use a compute instance with larger disk storage.
- Store temporary files in Azure Blob Storage instead of local disk.
- Implement data cleanup scripts to remove unnecessary files.

Lessons learnt:

- Monitor disk usage during long training jobs.
- Use cloud storage for large datasets instead of local disk.

---

43. Azure DevOps Fails to Publish Artifact Due to Exceeding Storage Quota

Problem description:
A pipeline fails when publishing build artifacts because the Azure DevOps storage quota is exceeded.

Error:

```
Error: Artifact upload failed. Insufficient storage space available.
```

What we need to analyze:

- The Azure DevOps storage usage under *Organization Settings* > *Usage*.
- The size of the artifacts being published.
- Whether unnecessary artifacts are being retained.

How to troubleshoot:

1. Check the storage usage in the Azure DevOps portal.
2. Review past artifacts to see if old ones can be deleted.
3. Inspect artifact retention policies to ensure unnecessary builds are not being kept.

How to resolve the issue:

- Delete old artifacts using:

```
Unset
az pipelines runs artifact delete --name <artifact-name>
```

- Reduce artifact size by compressing files before publishing.
- Increase the Azure DevOps storage quota if needed.

Lessons learnt:

- Regularly clean up old artifacts to avoid storage issues.
- Optimize artifact size before publishing to save space.

---

44. Azure Cognitive Services API Requests Failing Due to Rate Limits

Problem description:
API requests to Azure Cognitive Services (e.g., Computer Vision, Text Analytics) fail due to exceeding the allowed rate limit.

Error:
```
429 Too Many Requests - Rate limit exceeded
```

What we need to analyze:

- The Azure Cognitive Services pricing tier and rate limits.
- The number of requests being made per second.
- Logs to determine when the rate limit is being hit.

How to troubleshoot:

1. Check the Azure portal to view rate limits for your subscription tier.
2. Inspect logs to see when the requests exceed the threshold.
3. Use Application Insights to monitor API request patterns.

How to resolve the issue:

- Upgrade to a higher pricing tier with increased rate limits.
- Implement exponential backoff and retry logic in the application.
- Optimize API calls by batching requests where possible.

Lessons learnt:

- Monitor API usage and adjust pricing tiers accordingly.
- Implement retry mechanisms to handle transient failures.

45. Azure DevOps Pipeline Fails Due to YAML Syntax Errors

Problem description:
A pipeline fails to run due to incorrect YAML syntax in the pipeline configuration file.

Error:

```
Error: while parsing a block mapping, did not find expected key
```

What we need to analyze:

- The YAML file syntax.
- Indentation and incorrect spacing issues.
- Pipeline logs to identify the exact syntax error location.

How to troubleshoot:

1. Validate YAML syntax using an online YAML validator or command line:

```
Unset
yamllint azure-pipelines.yml
```

2. Check for missing colons, incorrect indentation, or misplaced line breaks.
3. Run a dry-run test using Azure DevOps pipeline validation.

How to resolve the issue:

- Correct YAML syntax errors based on the logs.
- Use consistent indentation (2 or 4 spaces).
- Refer to Azure DevOps YAML schema documentation for valid syntax.

Lessons learnt:

- Always validate YAML before committing changes.
- Use pipeline validation tools to catch syntax errors early.

---

46. Azure AI Model Deployment Fails Due to Unsupported Hardware SKU

Problem description:
A machine learning model deployment fails because the selected Azure compute instance does not support the required GPU or memory configuration.

Error:

```
Error: SKU 'Standard_NC6' not available in the selected region
```

What we need to analyze:

- The availability of the selected SKU in the region.
- The model's GPU/memory requirements.
- Alternative compute instance options.

How to troubleshoot:

1. Check Azure's SKU availability in the region using:

```
Unset
az vm list-skus --location eastus --output table
```

2. Validate the model's resource requirements against available SKUs.
3. Look for alternative regions that support the required SKU.

How to resolve the issue:

- Choose a different SKU that meets the model's requirements.
- Deploy the model in a region where the required SKU is available.
- Use Azure Machine Learning's auto-scaling feature to adjust resources dynamically.

Lessons learnt:

- Always check SKU availability before selecting a compute instance.
- Use regionally available resources to prevent deployment failures.

---

47. Azure Key Vault Access Denied in DevOps Pipeline

Problem description:
A DevOps pipeline fails to retrieve secrets from Azure Key Vault due to missing access permissions.

Error:
`Error: Access denied to Key Vault. Missing permissions for service principal.`

What we need to analyze:

- The Key Vault access policies and permissions.
- The service principal or managed identity used by the pipeline.
- Azure Active Directory (AAD) role assignments.

How to troubleshoot:

1. Check Key Vault access policies in the Azure portal.
2. Verify that the service principal has at least "Get" and "List" permissions for secrets.
3. Run the following command to check access:

```
Unset
az keyvault show --name <vault-name> --query "properties.accessPolicies"
```

How to resolve the issue:

- Assign the correct Key Vault permissions using:

```
Unset
az keyvault set-policy --name <vault-name> --spn <service-principal-id> --secret-permissions get list
```

- If using a managed identity, ensure it has the necessary Key Vault permissions.
- Use Azure RBAC instead of legacy Key Vault access policies for better security.

Lessons learnt:

- Ensure service principals and managed identities have proper access before deploying.
- Use role-based access control (RBAC) for better security management.

---

48. Azure DevOps Release Pipeline Fails Due to Agent Timeout

Problem description:
A release pipeline fails because the Azure DevOps agent times out before completing the task.

Error:

```
Error: The agent did not respond within the allotted timeout period.
```

What we need to analyze:

- The agent's resource utilization (CPU, memory).
- Network connectivity issues affecting the agent.
- Azure DevOps agent logs for timeout reasons.

How to troubleshoot:

1. Check agent logs in Azure DevOps (*Agent Pools* > *Agents* > *Logs*).
2. Increase the timeout setting in the pipeline YAML or UI.
3. Validate network stability between the agent and Azure DevOps.

How to resolve the issue:

- Use a self-hosted agent with higher compute resources if needed.
- Increase the task timeout in the pipeline YAML using:

```
Unset
timeoutInMinutes: 60
```

- Restart the agent and check for pending updates.

Lessons learnt:

- Monitor agent performance and scale resources accordingly.
- Optimize long-running tasks to prevent timeouts.

---

49. Azure AI Model Training Fails Due to Out-of-Memory Error

Problem description:
A deep learning model training job fails because it exceeds the available memory.

Error:
`RuntimeError: CUDA out of memory. Try reducing batch size.`

What we need to analyze:

- The model's batch size and memory usage.
- The GPU/CPU memory limits on the compute instance.
- Resource utilization logs during training.

How to troubleshoot:

1. Check available memory using:

```
Unset
nvidia-smi
```

2. Reduce batch size and check if training completes.
3. Monitor memory consumption in Azure ML logs.

How to resolve the issue:

- Reduce batch size in the training script:

```
Python
batch_size = 16  # Reduce from 32 or higher
```

- Use a compute instance with higher GPU memory (e.g., `Standard_ND40rs`).
- Implement gradient checkpointing to reduce memory usage.

Lessons learnt:

- Choose appropriate compute resources for deep learning workloads.
- Optimize memory usage by reducing batch size and using checkpointing.

---

50. Azure DevOps Fails to Deploy App Due to Locked Resource Group

Problem description:
A deployment fails because the target Azure resource group is locked for modifications.

Error:

```
Error: Resource group is locked and cannot be modified.
```

What we need to analyze:

- The lock status of the resource group in the Azure portal.
- Whether the lock is *Read-Only* or *Delete-Protected*.
- The purpose of the lock and whether it should be removed.

How to troubleshoot:

1. Check the lock in Azure:

```
Unset
az lock list --resource-group <rg-name>
```

2. Determine if the lock is needed for security purposes.
3. Verify who placed the lock and get approval for removal.

How to resolve the issue:

- Remove the lock if authorized:

```
Unset
az lock delete --name <lock-name> --resource-group <rg-name>
```

- If the lock is required, deploy resources in a different resource group.

Lessons learnt:

- Always check for locked resources before deploying.
- Use resource locks carefully to balance security and operational needs.

CAREER BYTE CODE
REALTIME PROJECTS PLATFORM

91 COUNTRIES 241k Learners
+32 471 40 89 08
CAREERBYTECODE.SUBSTACK.COM

→ TRAININGS

WE ARE DIFFERENT

At CareerByteCode, we redefine training by focusing on real-world, hands-on experience. Unlike traditional learning methods, we provide step-by-step implementation guides, 500+ real-time use cases, and industry-relevant projects across cutting-edge technologies like AWS, Azure, GCP, DevOps, AI, FullStack Development and more.

Our approach goes beyond theoretical knowledge—we offer expert mentorship, helping learners understand how to study effectively, close career gaps, and gain the practical skills that employers value.

16+
Years of operations

91+
Countries worldwide

241 K Happy clients

Our Usecases Platform
https://careerbytecode.substack.com

Our WebShop
https://careerbytecode.shop

CareerByteCode
Learning Made simple

CareerByteCode
All in One Platform

STAY IN TOUCH WITH US!

Website

Our WebShop https://careerbytecode.shop
Our Usecases Platform https://careerbytecode.substack.com

E-mail
careerbytec@gmail.com

Social Media
@careerbytecode

Phone
+32 471 40 8908

HQ address
Belgium, Europe

For any RealTime Handson Projects

And for more tips like this

Follow

Like & ReShare

@careerbytecode