# Resolving PolicyCortex Core Startup Errors in Real Data Mode

## Azure Credentials Configuration (Async Client Initialization Failure)

**Error:** *“⚠️ Failed to initialize async Azure client: No connection could be made because the target machine actively refused it. (os error 10061)”*[[1]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=Platform%20,policycortex_core%3A%20%E2%9C%85%20Fallback%20Azure%20client). This indicates the high-performance Azure client could not acquire credentials or reach Azure. A common cause is missing or incomplete Azure credentials, causing the DefaultAzureCredential chain to fall back to a method that failed (e.g. attempting Managed Identity or VS Code auth on a non-Azure machine, which was refused)[[1]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=Platform%20,policycortex_core%3A%20%E2%9C%85%20Fallback%20Azure%20client)[[2]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/AZURE_INTEGRATION_SUMMARY.md#L190-L198). In particular, the code requires an **Azure Tenant ID** and **Subscription ID** from environment variables, and a valid **Client ID/Secret** for a service principal if not using interactive login. If these are not provided, the async client will error out (it explicitly checks for AZURE\_TENANT\_ID)[[3]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client_async.rs#L139-L142).

**Fix:** Provide the necessary Azure credentials globally via environment or configuration, so the async client can authenticate without errors. In the .env or system environment, set the following (using your values):

* AZURE\_TENANT\_ID=<your-tenant-guid> – **(Required)** Tenant ID for your Azure AD (the code will error if missing)[[3]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client_async.rs#L139-L142).
* AZURE\_SUBSCRIPTION\_ID=<your-subscription-guid> – **(Required)** Subscription ID to target (the sync Azure client fails if missing)[[4]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L76-L84).
* AZURE\_CLIENT\_ID=<your-service-principal-app-id> – **(Required if using a service principal)** Client/App ID of the Azure AD application[[5]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/AZURE_INTEGRATION_SUMMARY.md#L106-L114).
* AZURE\_CLIENT\_SECRET=<your-service-principal-secret> – **(Required if using a service principal)** Client secret for the above app. (Store this securely; for development you can put it in your local .env, but never commit it.)
* Optionally, ensure az login is run if you prefer to use Azure CLI credentials instead of a client secret[[2]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/AZURE_INTEGRATION_SUMMARY.md#L190-L198).

With these in place, restart the core. The async Azure client should initialize successfully with a proper token (no “actively refused” error). If you want a code-level safeguard, you can add a check in AsyncAzureClient::new() to explicitly error out with a clear message when these env vars are not set, to avoid the obscure socket error. For example:

// In AsyncAzureClient::new(), before initializing credential:  
if std::env::var("AZURE\_CLIENT\_ID").is\_err() || std::env::var("AZURE\_CLIENT\_SECRET").is\_err() {  
 return Err("Azure Client ID/Secret not provided in environment".into());  
}

This would ensure missing credentials are caught early (similar to how tenant ID is checked)[[3]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client_async.rs#L139-L142). In summary, **the global fix is to supply valid Azure credentials** so the core runs in USE\_REAL\_DATA=true mode without falling back to any mock or local auth. Once configured, you should see the log message *“🚀 High-performance async Azure client initialized – ultra-fast data access enabled”* instead of the warning[[6]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L2-L10).

## Key Vault Secret for Database URL (Missing Secret Issue)

**Error:** *“Failed to get secret 'DATABASE\_URL' from Key Vault: retry policy expired and the request will no longer be retried”*[[7]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=2025,0%3A8080). This means the core tried to retrieve the database connection string from Azure Key Vault but could not find it or couldn’t access it. The SecretsManager connects to Key Vault (✅ Connected to Azure Key Vault: https://policycortex-kv.vault.azure.net/ confirms connectivity)[[8]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=2025,kv.vault.azure.net), then attempts get\_secret("DATABASE\_URL"). An absence of that secret (or insufficient access permissions) causes repeated failed attempts until the retry policy expires[[7]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=2025,0%3A8080). The code logs this warning and then falls back to check for a local env var[[9]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L116-L124). In our case, no secret was retrieved and likely no env var was set, resulting in an empty connection string.

**Fix:** Define the **DATABASE\_URL** secret so the application can obtain it. There are two approaches:

* **Store the secret in Azure Key Vault:** In the Key Vault named **policycortex-kv** (the default vault URL used in code[[10]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L48-L56)), create a secret with name **DATABASE\_URL**. Its value should be your PostgreSQL connection string (e.g. postgres://<user>:<pass>@<host>:5432/<db\_name>). Ensure the service principal or managed identity running PolicyCortex has **Get Secrets** permission on this vault. Once this secret is in place, the core will successfully retrieve it at startup (you’ll see a log *“Retrieved secret 'DATABASE\_URL' from Key Vault”* and no warning)[[11]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L96-L104)[[12]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L113-L118).
* **OR set an environment variable:** As a fallback, you can provide DATABASE\_URL in the environment (especially useful for local/dev runs). The SecretsManager code will use std::env::var("DATABASE\_URL") if the Key Vault fetch fails[[13]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L122-L129). In our log, after the KV miss, it ultimately got an empty string (meaning the env var was not set or empty)[[14]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=2025,core.exe). To fix this, set the DATABASE\_URL env var to the proper connection string so that even if Key Vault is empty, the core picks it up[[15]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L284-L292). For example, in a .env file:
* DATABASE\_URL=postgres://myuser:mypass@myserver.postgres.database.azure.com:5432/mydb

After providing the secret via one of the above methods, restart the app. The warning about the missing secret should disappear. The code will obtain a non-empty database\_url and proceed to initialize the DB pool with it[[15]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L284-L292).

## Database Connection Timeout (Postgres Pool Error)

**Error:** *“DB pool connection failed: pool timed out while waiting for an open connection”*[[14]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=2025,core.exe). This occurred because the database URL was not properly resolved (as noted above), so the application either tried to connect to an invalid/empty URL or the database was unreachable. The code attempts to connect if a database\_url string is present[[16]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L291-L300). In our case, an empty or incorrect connection string led to connection attempts that hung until timing out. Essentially, no database connection was established, yet the server continued startup (logging the failure as a warning)[[17]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L301-L309).

**Fix:** Ensure a **valid, reachable database** and connection string are provided. Once the DATABASE\_URL secret/env is fixed as above, verify that the URL is correct and the database server is running:

* If using Azure Database for PostgreSQL, confirm the server firewall allows connections from your environment (or disable public IP restrictions for testing). Update the URL with the correct host, username, password, and database name.
* If using a local Postgres for development, ensure it’s running and the URL (likely postgres://localhost/...) is correct.

When the app starts with a proper URL, you should see info!("Connected DB pool") in the logs[[18]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L295-L300), and the migration step will run (applying any SQLx migrations) without timing out. The pool timeout error will be resolved. In code terms, providing the correct connection string and network access is the “config change” needed – no code change is required other than possibly increasing the pool timeout if your DB is slow, but by default the error indicates it never connected at all. With a successful DB connection, the core can persist and query data as expected.

*(If you prefer a code safety net: you could modify the startup to panic or exit if database\_url ends up empty, rather than continuing. For example, after building database\_url, add if database\_url.is\_empty() { error!("No DB URL provided"); std::process::exit(1); }. This would prevent the app from running in a misconfigured state.)*

## Enabling Azure Cost Analytics and Policy Compliance Services

With the above errors fixed, the core will run using live Azure data. The next consideration is to ensure that **all Azure integrations are fully operational** (so that the application doesn’t silently fall back to mock data for certain features). Two key services that often need extra setup are **Cost Management (cost analytics)** and **Policy Insights (policy compliance)**. By default, the code will attempt to call Azure’s APIs for these when you hit the relevant endpoints (e.g. governance metrics, cost summaries, compliance status). To avoid runtime issues or empty data in those areas, make sure the following are in place:

* **Azure Cost Management API Access:** The service principal or identity must have permission to read cost data. If the subscription is under an Enterprise Agreement or billing account, assign **“Cost Management Reader”** or a higher role to the identity at that billing scope. For a single subscription context, being a Contributor/Owner on the subscription usually suffices[[19]](https://stackoverflow.com/questions/76392375/how-to-resolve-rbacaccessdenied-error-when-using-azure-api-to-query-billing-ac#:~:text=2,be%20assigned%20in%20this%20situation)[[20]](https://stackoverflow.com/questions/76392375/how-to-resolve-rbacaccessdenied-error-when-using-azure-api-to-query-billing-ac#:~:text=3,for%20Billing%20Account). The Cost Management resource provider (Microsoft.CostManagement) is typically registered by default for subscriptions[[21]](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/azure-services-resource-providers#:~:text=Learn%20learn,However%2C%20many), but if cost queries return authorization errors, double-check the role assignments. Once configured, the core’s cost queries (to the ARM CostManagement API) will succeed and populate cost metrics[[22]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L359-L368)[[23]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L399-L407).
* **Azure Policy Compliance (Policy Insights):** Ensure the **Microsoft.PolicyInsights** resource provider is registered in your subscription (this is required to query policy compliance data)[[24]](https://learn.microsoft.com/en-us/azure/governance/policy/assign-policy-template#:~:text=,that%20doesn%27t%20use%20managed%20disks). You can register it via Azure Portal (Subscriptions > Resource Providers > Register) or Azure CLI (az provider register --namespace Microsoft.PolicyInsights). Also, the service principal needs read access to Policy Insights data – in practice, a Subscription Reader or above covers this. If your subscription has no policy assignments yet, the compliance query may return no data; consider assigning at least one Policy or Initiative so that Policy Insights has something to report. With the provider enabled and proper RBAC, the core will be able to call the Policy Insights API to get compliance summaries[[25]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L184-L192). This surfaces as live compliance status and policy violation data in the APIs (instead of falling back to empty or simulated data).
* **Other Azure Services:** The application integrates with many Azure services (Activity Log, Resource Graph, Security/Defender, etc.). Most of these should work if the service principal has **Owner/Contributor** on the subscription (which grants access to Resource Graph, Monitor, etc.)[[2]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/AZURE_INTEGRATION_SUMMARY.md#L190-L198). If you encounter a 403 or error for a specific service, check if a special permission is needed. For example, listing Azure AD users (for the security IAM features) requires Graph API permissions (the app might need the **Directory.Read.All** application permission in Azure AD). Ensure to grant consent if using those features. Similarly, if using Defender alerts or other security center data, enable those services in Azure and give the app appropriate roles (Security Reader, etc.). The code is designed to log warnings if something is inaccessible, rather than crashing[[2]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/AZURE_INTEGRATION_SUMMARY.md#L190-L198), but the goal is to have no such warnings by configuring everything.

By configuring the above, you **enable the Cost Analytics and Policy Inspection features fully**. The core will operate in “Real Data” mode across the board, with no fallbacks to mock/simulated data. In summary, double-check the Azure setup: the service principal should have broad read permissions (or specific roles as needed), and required resource providers (like PolicyInsights) should be registered. This ensures that when the PolicyCortex Core starts and when you invoke various API endpoints, all integrations respond successfully with live data.

## Summary of Fixes

To run the Rust core backend with live Azure integrations and no fallbacks, apply these fixes:

* **Global Azure Config:** Set all required Azure env vars (AZURE\_TENANT\_ID, AZURE\_SUBSCRIPTION\_ID, AZURE\_CLIENT\_ID, AZURE\_CLIENT\_SECRET) so the async Azure client can authenticate without error[[3]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client_async.rs#L139-L142)[[4]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L76-L84). This resolves the initial credential error.
* **Database URL Secret:** Add the DATABASE\_URL secret to Key Vault **or** supply it via env, containing a valid Postgres connection string[[9]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L116-L124)[[15]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L284-L292). This fixes the secret-not-found warning and provides the core with DB connection info.
* **Database Connectivity:** Ensure the database is up and accessible with the provided URL. Once the core can connect (see “Connected DB pool” log)[[26]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L295-L303), the pool timeout error is resolved.
* **Azure Services Setup:** Grant the service principal **Reader/Contributor roles** on the subscription (and Cost Management Reader on billing if needed)[[19]](https://stackoverflow.com/questions/76392375/how-to-resolve-rbacaccessdenied-error-when-using-azure-api-to-query-billing-ac#:~:text=2,be%20assigned%20in%20this%20situation). Register the **PolicyInsights** provider[[24]](https://learn.microsoft.com/en-us/azure/governance/policy/assign-policy-template#:~:text=,that%20doesn%27t%20use%20managed%20disks) (and others if required). This prevents issues when fetching cost, compliance, and other Azure data.

By implementing the above (environment configuration changes and any needed code tweaks for clarity), you will eliminate the runtime errors seen in the log and enable full functionality. The PolicyCortex Core will run fully in live mode, initializing all Azure clients and services successfully and serving real Azure data (as intended by USE\_REAL\_DATA=true).

**Sources:** The fixes are based on the repository’s code and Azure integration documentation: - Environment variable requirements in AzureClient::new[[4]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L76-L84) and AsyncAzureClient::new[[3]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client_async.rs#L139-L142).  
- Secret retrieval logic in SecretsManager[[9]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L116-L124) and usage in core startup[[15]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L284-L292).  
- Database connection handling in main.rs[[27]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L295-L304).  
- Azure Policy and Cost API setup requirements from Microsoft Docs[[24]](https://learn.microsoft.com/en-us/azure/governance/policy/assign-policy-template#:~:text=,that%20doesn%27t%20use%20managed%20disks)[[21]](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/azure-services-resource-providers#:~:text=Learn%20learn,However%2C%20many) and project notes[[2]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/AZURE_INTEGRATION_SUMMARY.md#L190-L198).

[[1]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=Platform%20,policycortex_core%3A%20%E2%9C%85%20Fallback%20Azure%20client) [[7]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=2025,0%3A8080) [[8]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=2025,kv.vault.azure.net) [[14]](file://file-3zZbFxfNKvstK7ufc8NdUE#:~:text=2025,core.exe) corerunissues.md

<file://file-3zZbFxfNKvstK7ufc8NdUE>

[[2]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/AZURE_INTEGRATION_SUMMARY.md#L190-L198) [[5]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/AZURE_INTEGRATION_SUMMARY.md#L106-L114) AZURE\_INTEGRATION\_SUMMARY.md

<https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/AZURE_INTEGRATION_SUMMARY.md>

[[3]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client_async.rs#L139-L142) azure\_client\_async.rs

<https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client_async.rs>

[[4]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L76-L84) [[22]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L359-L368) [[23]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L399-L407) [[25]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs#L184-L192) azure\_client.rs

<https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/azure_client.rs>

[[6]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L2-L10) [[15]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L284-L292) [[16]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L291-L300) [[17]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L301-L309) [[18]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L295-L300) [[26]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L295-L303) [[27]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs#L295-L304) main.rs

<https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/main.rs>

[[9]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L116-L124) [[10]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L48-L56) [[11]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L96-L104) [[12]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L113-L118) [[13]](https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs#L122-L129) secrets.rs

<https://github.com/laeintel/policycortex/blob/8affb694ba944877233299b55d6de2580b577c24/core/src/secrets.rs>

[[19]](https://stackoverflow.com/questions/76392375/how-to-resolve-rbacaccessdenied-error-when-using-azure-api-to-query-billing-ac#:~:text=2,be%20assigned%20in%20this%20situation) [[20]](https://stackoverflow.com/questions/76392375/how-to-resolve-rbacaccessdenied-error-when-using-azure-api-to-query-billing-ac#:~:text=3,for%20Billing%20Account) How to resolve 'RBACAccessDenied' error when using Azure API to query billing account costs? - Stack Overflow

<https://stackoverflow.com/questions/76392375/how-to-resolve-rbacaccessdenied-error-when-using-azure-api-to-query-billing-ac>

[[21]](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/azure-services-resource-providers#:~:text=Learn%20learn,However%2C%20many) Find resource providers by Azure services - Microsoft Learn

<https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/azure-services-resource-providers>

[[24]](https://learn.microsoft.com/en-us/azure/governance/policy/assign-policy-template#:~:text=,that%20doesn%27t%20use%20managed%20disks) Quickstart: Create policy assignment using ARM template - Azure Policy | Microsoft Learn

<https://learn.microsoft.com/en-us/azure/governance/policy/assign-policy-template>