The 10 Most Common Azure Mistakes
(and how to fix them)

Slides at scottsauber.com

scottsauber

Audience

- Anyone using Azure
- Technically some of these translate to AWS too
- Hour long talk into 15 minutes – we are gonna go fast

Goals

- Ideas on how to make your Azure more maintainable, cost effective, and more performant
- At least one idea that you can go implement immediately at work

Who am I?

- Director of Engineering at Lean TECHniques
- Microsoft MVP
- Dometrain Author
- Redgate Community Ambassador
- Co-organizer of Iowa .NET User Group

Why this talk?

Account Organization

I mean, they're color coded for God's sake!

Problem: Account Organization is a mess

- One Subscription to rule them all
- Subscriptions named by app sometimes and by team other times
- Resource Groups with no real structure

Solution: Develop Account Organization Strategy

- Subscriptions - It Depends™
  - Minimum: per environment boundaries
  - Maximum: per team per environment boundaries

- Resource Groups – always per app per environment

- Service Groups – new as of May 2025
  - Fancy tags
  - Overlay to pull together across subscriptions, resource groups, etc – ie "Production apps" or "PCI Scope" or "MyApp"

Naming Standards

Pick a name, any name.

Problem: Naming is a mess

- Naming has no consistency across apps, teams, divisions

Solution: Implement a Naming Standard

- Microsoft has a [recommended naming convention](#) based on [recommended abbreviations](#)

- Example: app-navigator-dev-001 for an application called Navigator, that's an App Service running in the Dev env

- Azure has a [naming tool](#) you can run

- AzureNamingTool

- More important that you pick something and be consistent

- Enforce with Azure Policy (more soon)

Tagging Standards

TAG! YOU'RE IT!

Problem: Tagging is a mess

- Tags are metadata about the resource

- No consistent tags

- Makes it hard to answer questions like:
  - How much does this app cost me a month?
  - How much does the Dev environment cost me a month?`
  - Who owns this resource?

🔍 Search ◇ «

☑ Delete all 👥 Feedback

📘 Overview

🔲 Activity log

👥 Access control (IAM)

◆ Tags

☁ Quick start

🔧 Diagnose and solve problems

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. Tag names are case insensitive, but tag values are case sensitive. Learn more about tags⧉

Do not enter names or values that could make your resources less secure or that contain personal/sensitive information because tag data will be replicated globally.

Name ⓘ   **Value** ⓘ

[   ] : [   ]

Solution: Implement a Tagging Standard

- Define a common set of Tags to be used on each resource

- Common tags:
  - Environment (Dev, Test, Production)
  - Owner (so.andso@contoso.com)
  - BusinessUnit (Finance, Sales, etc)
  - Application (CRM, PaymentsPortal, etc)
  - CostCenter (abc123, etc)
  - ServiceClass (Tier1, Tier2, etc)

- Enforce with Azure Policy (more soon)

Azure Policies

We have a strict policy here - not to do that.
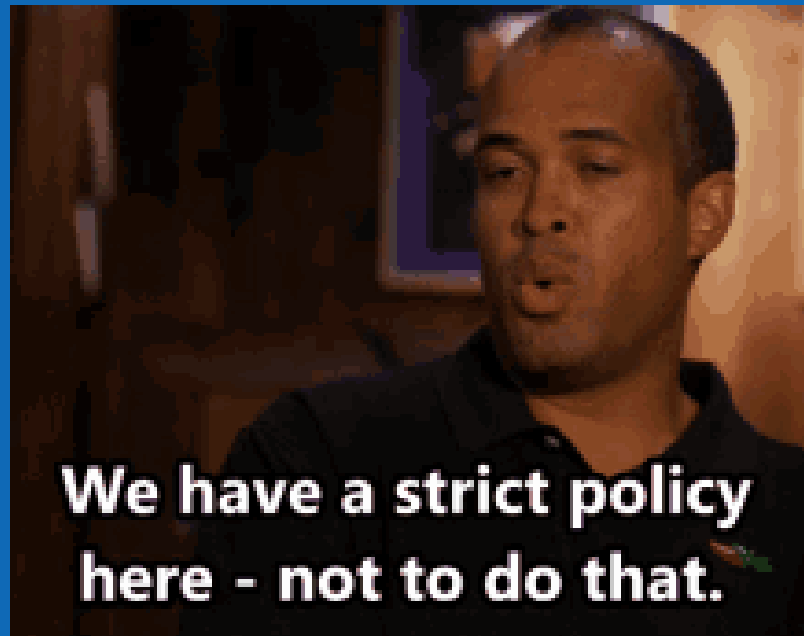
Problem: We aren't enforcing our standards

- You have a document defining standards
- But it's not followed

Solution: Azure Policies

- Azure Policies can enforce your naming standards

- And your tagging standards

- Can have multiple Policy Effects:
  - Deny resources from being created - PREFERRED
  - Audit - flag it as non-compliant
  - Modify – auto change it – DON'T DO THIS

- Can ensure resources are configured correctly too
  - Ensure HTTPS enforcement
  - SQL has Data Encryption at rest turned on

Managed Identities

Problem: We are managing credentials

- Credentials to manage – create, store, rotate, revoke
- Azure SQL still using SQL Auth – username + password
- Key Vault still using Client ID + Client Secret
- Someone might accidentally commit a secret

Solution: Use Managed Identities

- Managed Identities allow you to give a resource an "identity" (user to run as essentially)

- These Identities are managed completely by Azure – no username/password/client id/client secret needed

- Assign a Managed Identity to an App Service/Function/etc

- Give that Managed Identity access to read secrets, connect to a DB, etc

- Completely passwordless

- In your code you can use DefaultAzureCredential to do the right thing (more on this later)

DefaultAzureCredential

Problem: People use DefaultAzureCredential…

- …without knowing the tradeoffs
- DefaultAzureCredential is meant to get you into Azure as quickly as possible
- The downside is – performance

Problem: People use DefaultAzureCredential...

CREDENTIALS

Environment → Workload Identity → Managed Identity → Visual Studio → Azure CLI → Azure PowerShell → Azure Developer CLI → Interactive browser

CREDENTIAL TYPES

Deployed service   Developer tool   Interactive

Solution: Be explicit about credentials

- Locally – use AzureCli or VisualStudioCredential… else use ManagedIdentity

- … or use Excludes on DefaultAzureCredential

- This will save you ~2-10 seconds when interacting with the credential

- DefaultAzureCredential Best Practices

- My blog post on this in 2022

Solution: Be explicit about credentials

```
// Old
builder.Configuration.AddAzureKeyVault(
    new Uri($"https://kv-myapp-{builder.Environment.EnvironmentName}.vault.azure.net"),
    new DefaultAzureCredential());

// New
TokenCredential tokenCredential;
if (builder.Environment.IsEnvironment("Local"))
{
    tokenCredential = new ChainedTokenCredential( params sources: new AzureCliCredential(),
                new VisualStudioCredential());
}
else
{
    tokenCredential = new ManagedIdentityCredential();
}
builder.Configuration.AddAzureKeyVault(
    new Uri($"https://kv-myapp-{builder.Environment.EnvironmentName}.vault.azure.net"),
    tokenCredential);
```

Requiring TLS 1.2+

Problem: Not requiring TLS 1.2+

- HTTPS has underlying encryption algorithms to encrypt your data in transit

- These encryption algorithms have continually gotten better and better

- Not requiring TLS 1.2+ allows users to connect to your app with insecure protocols
  - Example – people on older browsers

- This allows a man-in-the-middle attacks for people to decrypt in real time, exposing your data

Solution: Require TLS 1.2+

- Require TLS 1.2+ on all your relevant resources (ie app service, function apps, storage accounts, etc)

- Make sure you also enforce a good cipher suite
  - ie TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 and TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

- TLS 1.3 is the latest, most browsers in the last 4-5 years support 1.3

- SSL Labs to test your site - https://www.ssllabs.com/ssltest/

FederatedCredentials

Problem: Not using Federated Credentials

- Your CI/CD pipelines need to auth to Azure to provision resources, deploy, etc

- Without Federated Credentials, you're managing a Client ID + Client Secret (most likely)

- All the same issues managing credentials (create, store, rotate, revoke)

- If someone gets these credentials, they can manipulate your Azure environment

Solution: Use Federated Credentials

- Federated Credentials allows you to "trust" repositories to push to Azure

- Passwordless

- Short-lived tokens (minutes)

- [How to do this with GitHub Actions](#)

Configure an Microsoft Entra ID managed identity or an identity from an external OpenID Connect Provider to get tokens as this application and access Azure resources.

Federated credential scenario *   GitHub Actions deploying Azure resources   ⌄

Connect your GitHub account

Please enter the details of your GitHub Actions workflow that you want to connect with Microsoft Entra ID. These values will be used by Microsoft Entra ID to validate the connection and should match your GitHub OIDC configuration. Issuer has a limit of 600 characters. Subject Identifier is a calculated field with a 600 character limit.

Issuer ⓘ   https://token.actions.githubusercontent.com
                Edit (optional)

Organization * scottsauber

Repository *   workshop-dotnet-azure-github-bicep

Entity type *   Branch   ⌄

Based on selection *   main

Budgets and Cost Alerts

Problem: Cost isn't managed in Azure

- Sudden, unexpected bills
- No early warning for forecast overrun
- Lack of visibility across teams
- Missed anomalies and fraud detection

Solution: Use Azure Budgets

- Define a spending threshold for a scope (ie Subscription or Resource Group)

- Tracks actual or forecasted against that target over a time period (monthly, quarterly, yearly)

- Can have multiple thresholds, 50%, 80%, 100%

- Can trigger Action Groups (email, Functions, Webhooks)

- Budgets are a bad name – does not stop resources, only monitoring + triggers

- Can lag by 24 hours

Solution: Use Azure Cost Alerts

- ## Anomaly Detection

Anomaly alert: An unusual cost increase was detected

An unusual cost increase was detected on 8/8/2025 12:00:00 AM for the sub-sandbox subscription. Cost Management detected a possible cost anomaly based on daily cost trends between 6/10/2025 12:00:00 AM and 8/7/2025 12:00:00 AM. Please review changes to determine whether this was expected.

Subscription summary

Anomaly detected	Yes
Delta compared to expected range	74.99 %

Resource group summary

- Cost changed 18.9% from 9 existing resource group(s).

Most significant changes in resource group(s) during this period

Name	Cost change %	Percent of total
rg-gorman-sandbox-ncus-dev	104.57	11.22
rg-documenttranslation	346.34	7.73
rg-alec-rg-aisearch	-0.36	0.05

Review additional details in the Azure portal.

Details >

Azure Reservations

Problem: Cost is getting too high

- Cloud isn't always cheaper up front
- Sometimes cloud is about speed, lower TCO, and more visibility

Azure Savings Plans

OOOOOOOWWWEEEE!

Problem: Cost is getting too high

- Cloud isn't always cheaper up front
- Sometimes cloud is about speed, lower TCO, and visibility

Solution: Azure Savings Plans

- Commit to using certain spend $ for 1 yr or 3 yrs, get a discount
- Can be used however you want across eligible resource types

Comparing Savings Plans vs Reservations

- Reservations give you more savings, but more risk if you need to change your plans

- I recommend everyone use at least Savings Plans

- Go tell your boss you can cut the Azure bill by at least 20% and get a promotion

- Comparison of a single P0V3 App Service Plan:

	Pay As You Go	Savings Plan	Reservation
1 year	394.96 kr./mo	269.76 kr./mo (32%)	235.00 kr./mo (41%)
3 year	394.96 kr./mo	197.83 kr./mo (50%)	162.69 kr./mo (59%)

Real benefits of these practices

- 1 company went from a request lead time of 1 month for new infrastructure to ~30 minutes

- Another company reduced their overall Azure cost by almost 80%

- Reduced need of manual security provisioning due to Azure account organization cascading rules

- Increased security posture

- Improved application performance

scottsauber

Takeaways

- Hopefully you got at least one idea today you can take back to work
- This slide deck is your resource – up at scottsauber.com

scottsauber

Questions?

ssauber@leantechniques.com
@scottsauber.com on Bluesky
@scottsauber on Twitter
/scottsauber on LinkedIn

Book free time with me:

Slides at scottsauber.com

scottsauber

Thanks!

scottsauber