

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

# Audience

- Anyone using Azure
- Technically some of these translate to AWS/GCP too

# Poll

- Who's a ~~Developer~~ Prompt Engineer?
- Who's a DevOps/Cloud/Platform Engineer?
- Who's responsible for their Azure environment?

# Agenda

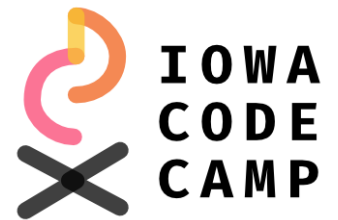
- Series of lightning talks
- Account Organization
- Naming Standards
- Tagging Standards
- Azure Policies
- Managed Identities
- DefaultAzureCredential
- Requiring TLS 1.2+
- Federated Credentials
- Budgets and Cost Alerting
- Reservations
- Savings Plans
- ++ AI topics

# 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](#) and [Iowa Code Camp](#)



# Why this talk?



# 2026 version



# 2026 version – but with lifting

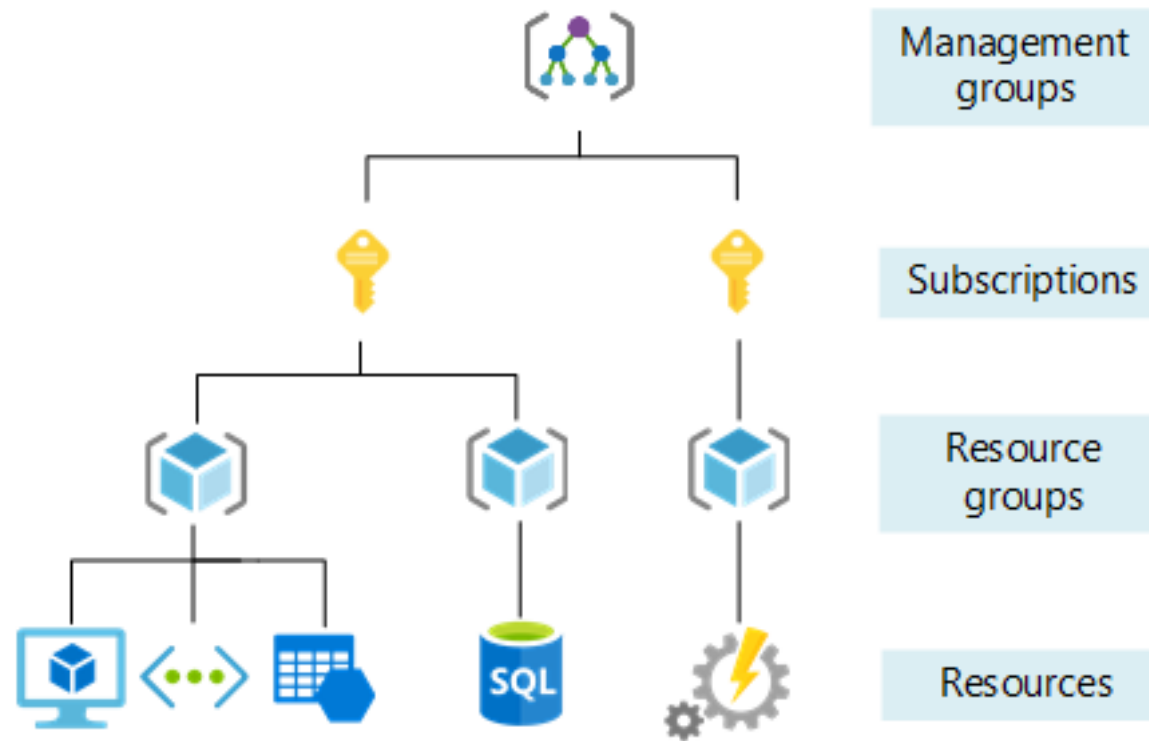


# Account Organization



# Subscriptions vs Resource Groups

- Ignoring Management Groups bc you probably don't care about those



# 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”
  - Still in Preview

# Naming Standards



# 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
- Storage accounts don't allow dashes 😞
- Enforce with Azure Policy (more soon)

# Tagging Standards



# 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?

◇ <<  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](#)

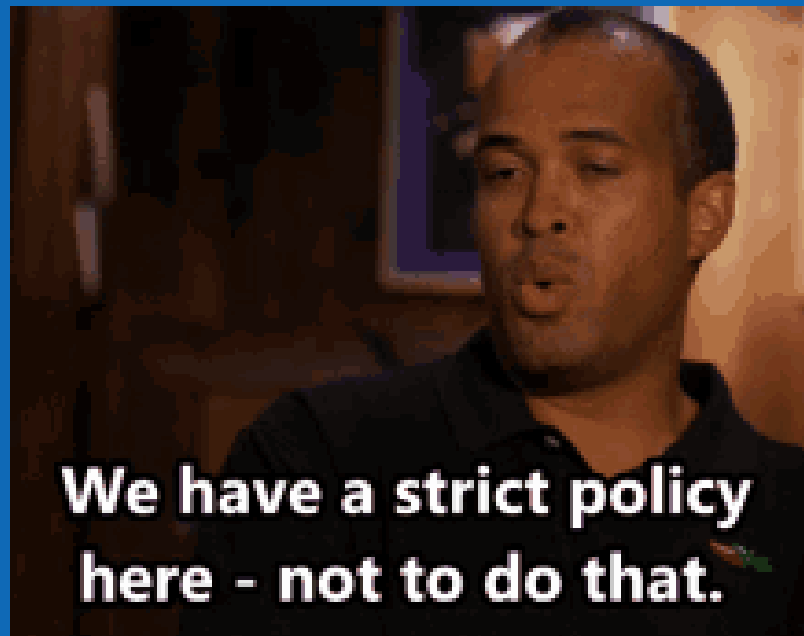
Do not enter names or values that could make your resources less secure or that contain personal/sensitive information because tag data will be replicated globally.

| Name ⓘ               | Value ⓘ              |
|----------------------|----------------------|
| <input type="text"/> | <input type="text"/> |

# Solution: Implement a Tagging Standard

- Define a common set of Tags to be used on each resource
- Common tags:
  - Environment (Dev, Test, Production)
  - Owner (team.name@contoso.com)
  - BusinessUnit (Finance, Sales, etc)
  - Application (CRM, PaymentsPortal, etc)
  - CostCenter (abc123, etc)
  - ServiceClass (Tier1, Tier2, etc)
- Enforce with Azure Policy (more next)

# Azure Policies



# 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
- Azure Blob Storage Account access keys
- Azure Service Bus using connection strings
- Someone might accidentally commit a secret
- Agents read local secret files
  - Magic local .gitignored files were an anti-pattern before AI, worse now

# 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)

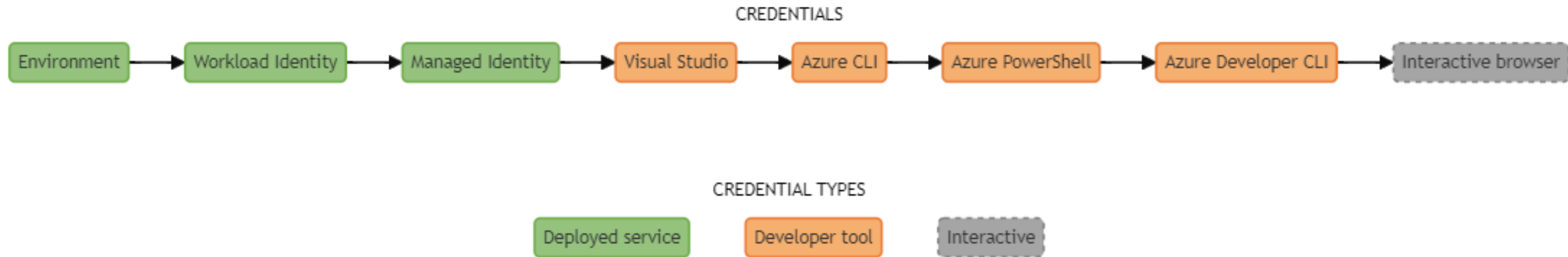
# DefaultAzureCredential



# 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...



# 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+



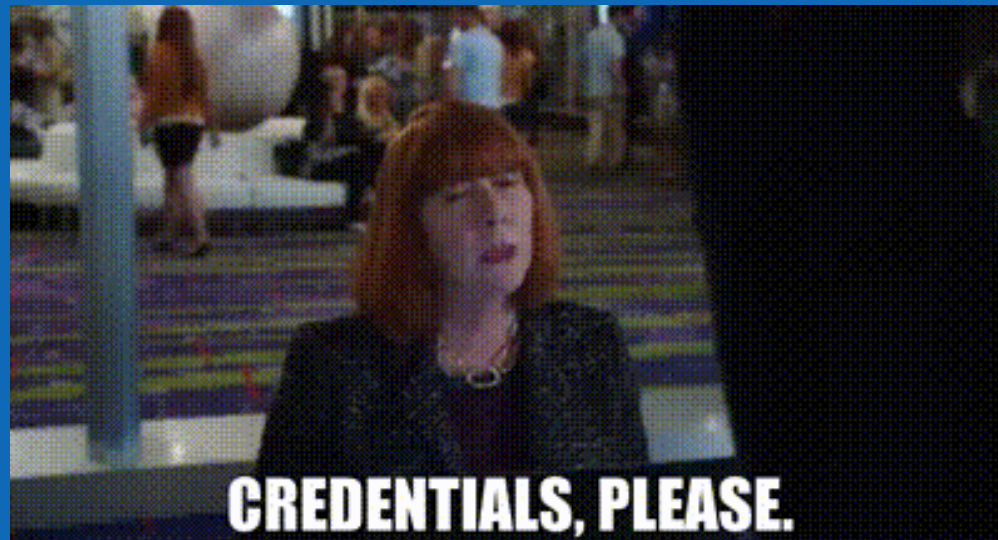
# Problem: Not requiring TLS 1.2+

- HTTPS has underlying encryption algorithms to encrypt your data in transit
- These encryption algorithms have continually gotten better and better
- Hardware has gotten better and better to crack old algorithms
- 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+

- Newer resource provisioning defaults to 1.2+, but existing deployed services you have don't automatically change
- 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 8 years support 1.3
- SSL Labs to test your site - <https://www.ssllabs.com/ssltest/>
- Enforce with Azure Policy

# 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](#)
- Prefer User Managed Identity over App Registration

# Solution: Use Federated Credentials

Federated credential scenario \* ⓘ Configure a GitHub issued token to impersonate this managed identity and deploy to Azure ▼

[Configuration guide for GitHub identities](#) 🔗

ⓘ To ensure secure FIC access for GitHub Actions, use the new subject identifier format with immutable organization and repository IDs. [Learn more](#)

## 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 URL \* ⓘ https://token.actions.githubusercontent.com

[Edit \(optional\)](#)

Organization \* GitHub organization name

Organization ID \* ⓘ GitHub organization ID

Repository \* GitHub repository name

Repository ID \* ⓘ GitHub repository ID

Entity \* Select ▼

Subject identifier ⓘ repo:{Organization}@{Organization ID}/{Repository}@{Repository ID}:{Entity}

This value is generated based on the GitHub account details provided. [Edit \(optional\)](#)

# Solution: Use Federated Credentials

Entity \*

Subject identifier ⓘ

## Credential details

Enter and review the details for this credential

Select

Environment

Branch

Pull Request

Tag

# GitHub Integration First Class

Federated credential scenario \* ⓘ

ⓘ To ensure secure FIC access for GitHub Actions, please use the GitHub Actions scenario.

## Connect your GitHub account

Please enter the details of your GitHub Actions workflow and the Entra ID to validate the connection and show the GitHub Actions workflow.

Issuer URL \* ⓘ

Configure a GitHub issued token to impersonate this managed identity and deploy to Azure

**GitHub Actions deploying Azure resources**  
Configure a GitHub issued token to impersonate this managed identity and deploy to Azure

**Kubernetes accessing Azure resources**  
Configure a Kubernetes service account to get tokens as this managed identity and access Azure resources

**Other**  
Configure an identity managed by an external OpenID Connect Provider to access Azure resources as this managed identity

[Edit \(optional\)](#)

# 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-airesearch      | -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

# Solution: Azure Reservations

- Commit to using certain resources for 1 year or 3 years, get a discount between 20% - 70% depending on the resource (App Service Plans, SQL, VMs, Cosmos, etc)
- Downside – what if you get rid of the resource after 2 years but you committed to 3
- Exchanges are allowed – go sideways or up
- Still pay monthly or up front (if you want)

# Azure Savings Plans



# 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 P0V4 App Service Plan:

|        | Pay As You Go | Savings Plan       | Reservation        |
|--------|---------------|--------------------|--------------------|
| 1 year | kr611.967/mo  | kr422.244/mo (31%) | kr360.979/mo (41%) |
| 3 year | kr611.967/mo  | kr281.519/mo (54%) | kr232.589/mo (62%) |

# AI Coding Agent Tips for Azure



# Problem: Need Azure Context

- Coding agents know about your codebase by default
- They don't know about your Azure environment

## Solution: az CLI

- Azure CLI already exists to get you access to everything in Azure
- Install and then `az login`
- Don't even need to tell your agent to do this, just login and then it'll self-discover
- Risk: it runs as you – are you a global admin?
  - This was a problem before AI
- Fix: new user that's got Reader only that you `az login` as
- Additional Fix: auto mode in Claude is 8x more likely to catch dangerous actions than a human approving

## Solution: Then what with the agent + Azure?

- Have it review existing infrastructure (best practices, security, cost management evaluation, etc)
- Have it review the infrastructure and generate IAC for new resources
- Have it pull your error logs for you
- Audit RBAC assignments
- Have it troubleshoot production issues (is it CPU, memory, network, disk, etc)
- Troubleshoot: take in the context of your codebase, the deployments, and how it's running in production

# Problem: I have error logs, but manually review

- Error logs sitting in Log Analytics Workspace or App Insights
- Maybe you've got alerting set up, maybe not
- Requires a human to see that alert, review the error, and send the fix
- What if...

**A human sees the alert**

**A human reviews the error**

**A human sends the fix**

**Another human reviews the fix**

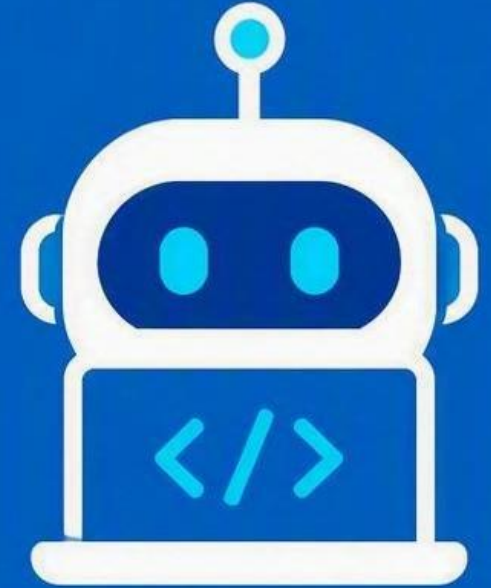
**An AI sees the alert**  
**An AI reviews the error**  
**An AI sends the fix**  
**A human reviews the fix**



**Azure Log Analytics**



**GitHub Issue**



**GitHub Copilot Coding Agent**

## Final Tip:

- How do we automate more of the SDLC beyond coding?

# 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](http://scottsauber.com)

# Questions?

ssauber@leantechniques.com  
@scottsauer.com on Bluesky  
@scottsauer on Twitter  
LinkedIn 📌

Add me on LinkedIn:



# Thanks!