

Azure Naming Tool Overview

Version	Date	Author	Notes
1.0	11/20/2024	Kevin Sebastian	Detailed overview of azure naming tool

1. Business Objective/Summary

Purpose

The Azure Naming Tool API is designed to provide a streamlined and automated way for pipelines and users to generate resource names that align with established naming conventions in Azure and follow Insight Global's naming standards. This API plays a critical role in ensuring consistent naming standards across the organization's cloud environment, reducing human errors and enhancing governance.

Business Objective

In cloud environments, maintaining consistent naming conventions is essential for resource management, security compliance, and operational efficiency. The API is a central component that integrates seamlessly into CI/CD pipelines, automating the process of name generation based on pre-defined rules. By using the Azure Naming Tool API, organizations can standardize the naming of resources, ensuring that all deployed assets adhere to company policies and best practices.

Key Benefits

- *Improved Compliance:* Enforces adherence to organizational naming policies, reducing the risk of mismanagement or compliance issues.
- *Efficiency and Scalability:* Automates the name generation process, saving time and reducing manual errors.
- *Enhanced Integration:* Easily callable from various pipelines and tools, supporting DevOps practices and continuous deployment workflows.
- *Centralized Configuration:* Offers a single source of truth for naming conventions, managed through a user-friendly interface or programmatically through the API.
- *Broader System Fit:* The Azure Naming Tool API fits into the broader cloud management and DevOps ecosystem by acting as a crucial compliance and governance layer within resource deployment processes. It supports IT administrators and development teams by simplifying the implementation of naming conventions across different projects, regions, and teams.

Problem Solved

The primary problem addressed by this API is the inconsistency in naming Azure resources, which can lead to difficulties in managing cloud environments, enforcing policies, and maintaining clear and organized resources. By integrating the API into pipelines, organizations ensure uniformity, aiding in resource tracking, cost management, and policy enforcement.

Target Audience

- **Developers:** Individuals who integrate and call the API within their CI/CD pipelines or software systems.
- **Cloud Administrators:** Professionals responsible for managing and maintaining Azure environments who need to ensure compliance with naming standards.
- **DevOps Engineers:** Teams focused on automating deployment pipelines and maintaining continuous integration and delivery practices.
- **IT Operations Teams:** Staff responsible for overseeing cloud resources and ensuring they are organized, discoverable, and compliant with company policies.
- **Project Managers:** Those who require an understanding of the tool's functionality and benefits for project planning and resource allocation.

2. Project Directory Structure Overview

Subdirectories

1. **archive**
 - *README.md*, *v1.zip*: Documentation and version archive, indicating previous iterations or releases of the project.
2. **Attributes**
 - Contains attribute classes like *ApiKeyAttribute.cs*, *CustomHeaderSwaggerAttribute.cs*, and *RateFilter.cs*, which help enforce custom behavior across API calls and routes.
3. **Components**: Main directory for Blazor components.
 - *App.razor*, *Routes.razor*: Core files for component registration and routing.
 - *General*, *Instructions*, *Layout*, *Modals*, and *Pages* subfolders contain:
 - *General*: Components like *AnchorNavigation.razor* and *ExternalContent.razor* for UI enhancements.
 - *Instructions*: User guidance components (e.g., *AdminConfigurationInstructions.razor*).
 - *Layout*: Defines the main app structure with *MainLayout.razor* and the navigation menu *NavMenu.razor*.
 - *Modals*: UI modals such as *AddModal.razor* and *EditModal.razor*.
 - *Pages*: Main UI pages (e.g., *Index.razor*, *Admin.razor*, *Generate.razor*).
4. **Controllers**
 - Controllers for handling HTTP requests, including *AdminController.cs*, *CustomComponentsController.cs*, and *PolicyController.cs*. These enable the backend logic for managing admin functions, custom components, and policy settings.
5. **Deployments**
 - *AppService-WebApp*: Contains *solution.bicep* and JSON files for deploying the application to an Azure App Service.
 - *modules*: Subdirectory with Bicep modules for infrastructure deployment (e.g., *rgWebApp.bicep*).
6. **Helpers**: Utility classes for various functionalities:
 - *CacheHelper.cs* and *FileSystemHelper.cs*: Handle caching and file operations.
 - *GeneralHelper.cs* and *LogHelper.cs*: Provide common utility functions and logging mechanisms.
7. **Models**
 - Defines data structures such as *AdminUser.cs*, *ResourceComponent.cs*, and *ValidateNameRequest.cs*. Includes a subfolder *PolicyDefinition* for policy-related classes (e.g., *PolicyRule.cs*).
8. **Properties**
 - Contains *launchSettings.json* for local development environment settings.
9. **repository**
 - JSON configuration files representing stored application data (e.g., *adminusers.json*, *resourceorgs.json*).
10. **Services**
 - Business logic classes like *AdminService.cs*, *CustomComponentService.cs*, and *PolicyService.cs*, supporting data operations and process management.
11. **settings**
 - Includes placeholder files like *blank.txt*.
12. **wwwroot**: Publicly accessible static assets:
 - *app.css*, *favicon.ico*, *favicon.png*: Styling and icons.
 - *css* folder: Contains site styles and *bootstrap* framework files.

- *images*: Logos and images for UI elements.
- *js*: JavaScript files, including *bootstrap* and *jquery*.

13. Project Root

- *.dockerignore*, *Dockerfile*: Configuration files related to Docker for containerization.
- *AzureNamingTool.csproj*, *AzureNamingTool.sln*: Project and solution files for the .NET application.
- *Program.cs*: The main entry point of the application.
- *appsettings.json*, *appsettings.Development.json*: Configuration files for environment settings.
- *configurationfileversions.json*, *programsettings.json*, *versionalerts.json*: Additional configuration files for versioning and program-specific settings.
- *connectiontest.png*: An image likely used for documentation or testing purposes.
- *samplewebhookdata.json*: Sample data for webhook testing.

Directory Purpose Summary

- *Attributes*, *Components*, *Controllers*, *Helpers*, *Models*, *Services*: Primary code structure supporting backend logic and front-end Blazor components.
- *Deployments*, *repository*, *settings*: Configuration and deployment details.
- *wwwroot*: Static resources, styling, and documentation assets.

Potential Changes

1. **UI/Frontend Changes**: For UI changes, modifications would typically occur in the *Components\Pages* directory. For instance, if you need to update the homepage, edit the *Index.razor* file. If adding a new page such as an "Admin Dashboard," you would create a new Razor component (e.g., *AdminDashboard.razor*) within *Pages*. For changes in the layout or navigation structure, you would modify *NavMenu.razor* or *MainLayout.razor* in the *Components\Layout* directory. For new modals like "User Settings," add a component in *Components\Modals* (e.g., *UserSettingsModal.razor*).
2. **Backend Changes (Business Logic)**: For changes to business logic, you would generally update or add new services in the *Services* directory. For example, if a new service is required to handle data processing for an admin function, you would create or modify a service like *AdminService.cs*. If the logic impacts a specific component or resource, such as a new component for handling user roles, updates should be made to files like *RoleManagementService.cs*.
3. **API Changes**: When adding or modifying API endpoints, you need to update or create new controller classes in the *Controllers* directory. For example, to add a new API endpoint for handling custom components, create a new file like *CustomComponentsController.cs*. If any new API-specific logic is required, modify the corresponding service in the *Services* directory to process the data. For integrating new routes, you may need to modify the *App.razor* or *Routes.razor* file in *Components*.
4. **Data Validation and Processing Changes**: If you need to introduce new validation logic for incoming data or ensure specific fields meet certain criteria, changes should be made in the *Helpers* directory, particularly in files like *ValidationHelper.cs*. If there's complex data processing that affects multiple modules, consider updating *GeneralHelper.cs* or creating a new helper class (e.g., *DataProcessingHelper.cs*) in *Helpers*.
5. **Modifying or Adding New Data Models**: When creating or updating data models, such as introducing a new resource for components, you'll need to modify or create a new class in the *Models* directory. For example, if a new data model for tracking "ComponentLogs" is required, create a file like *ComponentLog.cs* within the *Models* folder. If you need to add policy-related data structures, modify or extend the *PolicyDefinition* models in the *Models\PolicyDefinition* directory, such as adding a new *PolicyRuleFactory.cs*.

6. **Configuration and Settings Changes:** For any changes related to configuration, such as adding new settings for a feature like logging or API rate limiting, you'd update the *appsettings.json* or *appsettings.Development.json* files. Additionally, to reflect these changes in the repository or other configurations, update the corresponding files in the *repository* folder, such as *appsettings.json* or *generatednames.json*. If new settings are required for deployment, modify *launchSettings.json* found in the *Properties* folder.
7. **Infrastructure and Deployment Changes:** If you're making changes to infrastructure, such as deploying a new service or modifying existing ones (e.g., Azure App Service), you'd update the Bicep or JSON files in *Deployments\AppService-WebApp*. This could involve creating new modules like *appService.bicep* or modifying deployment configurations in *solution.bicep* to ensure the new infrastructure aligns with the project requirements. If there's a need for a new environment or configuration, add those specifications in the *modules* subdirectory.
8. **New Feature Implementation (Example: Webhook Integration):** For a new feature like webhook integration, create a new controller in *Controllers*, such as *WebhookController.cs*, to handle incoming webhook requests. Supporting backend logic should be added to *Services* by creating a new service like *WebhookService.cs*. Data models to define incoming/outgoing webhook data would be added in *Models*, such as *WebhookRequest.cs* and *WebhookResponse.cs*. For UI updates, new Razor pages or modals may be added in *Components\Pages* or *Components\Modals* (e.g., *WebhookConfiguration.razor*). Configuration values related to the webhook can be added in *appsettings.json*, while any needed data for the webhook could be stored in the *repository* folder.
9. **Adding or Updating Static Content (Images, CSS, JS):** For static content changes, such as adding a new logo or updating the CSS for a new theme, modify or add files in the *wwwroot* folder. For CSS, the *wwwroot\css* folder contains styles like *site.css*, which should be updated. If you are adding new images, such as for a promotional banner or logo, add them to the *wwwroot\images* directory. For JavaScript-related updates, particularly for new frontend functionality like interactive forms or new Bootstrap components, modify or add files in the *wwwroot\js* directory, such as *bootstrap.bundle.js*.
10. **Documentation and Screenshots Updates:** If you need to add new documentation or update existing visual documentation, place updated screenshots or new documentation in the *wwwroot\Screenshots* directory. For example, if you've added a new feature, like a user authentication process, a screenshot showing the new workflow should be added in *wwwroot\Screenshots* (e.g., *UserAuthentication.png*). If you're updating any textual documentation or readme files for new features or changes, place them in the *archive* folder or update files like *README.md*.
11. **Logging and Monitoring Changes:** For changes related to logging, such as introducing a new logging framework or integrating Azure Monitoring, the *Helpers* directory would likely need to be updated, specifically within *LogHelper.cs* to manage logging levels and output formatting. Additionally, configuration for logging might need to be added to *appsettings.json* or *programsettings.json* to enable or adjust logging verbosity or destinations (e.g., sending logs to Azure Log Analytics). If the monitoring includes new alert mechanisms, files like *versionalerts.json* in the *repository* folder should be updated.

3. Key Components and Features

API's primary components

A detailed explanation of the API's primary components involves outlining the different parts of the application and their roles in its functionality. Here's a breakdown based on the directory structure you've provided, applying it to an example API:

API Routes (Controllers)

Location: *Controllers* Directory

1. **Primary Role:** API routes are defined in controller classes, where incoming HTTP requests are mapped to specific actions (methods). These actions determine how the requests are processed, what data is retrieved or modified, and what responses are returned.
2. **Components:**
 - **AdminController.cs:** Handles requests related to admin functions, such as viewing logs, managing configurations, and user actions.
 - **CustomComponentsController.cs:** Handles operations for custom components like creating, updating, and retrieving data related to custom components.
 - **PolicyController.cs:** Manages API routes related to policy creation, updates, and retrieval.
 - **Resource*Controller.cs** (e.g., *ResourceLocationsController.cs*, *ResourceTypesController.cs*): Handle requests related to specific resource types, including CRUD (Create, Read, Update, Delete) operations on resources.
3. **Role in API Functionality:** Each controller is responsible for handling specific requests and delegating actions to the corresponding services. For example, the *ResourceLocationsController.cs* would handle requests to manage resource locations, with the service determining the logic for these actions (retrieving, creating, or modifying resource data).

Data Models

Location: *Models* Directory

1. **Primary Role:** Data models define the structure of the data that is transferred between the client and the API. They represent the data entities (objects) used in API requests and responses.
2. **Components:**
 - **ResourceLocation.cs:** Defines the data structure for resource locations, including attributes like location name, address, and type.
 - **GeneratedName.cs:** Represents the generated names for resources based on naming conventions, including attributes such as name and status.
 - **PolicyDefinition.cs:** Contains data relevant to the policy, including its name, type, and rules.
3. **Role in API Functionality:** Models ensure that the data sent and received through the API is consistent. For example, the *GeneratedName.cs* model is used when creating or retrieving names for resources, ensuring that all responses are properly formatted and follow the correct structure.

Data Processing Services

Location: *Services* Directory

1. **Primary Role:** Services contain the business logic that processes data and interacts with databases, external services, or other systems. They handle the core functionality of the API, such as validation, data manipulation, and business rules.
2. **Components:**
 - **AdminLogService.cs:** Manages logging actions, including writing logs when users perform administrative actions.
 - **ResourceLocationService.cs:** Handles the logic for resource location management, including creating, reading, and updating resource locations.

- **PolicyService.cs:** Contains logic for managing policies, including creating new policies, validating policy rules, and retrieving policy details.
3. **Role in API Functionality:** Services abstract the core logic and interact with data models and repositories. For instance, *ResourceLocationService.cs* would handle the logic for validating and storing new locations when a request is made to the *ResourceLocationsController.cs*.

Database Repositories

Location: *repository* Directory

1. **Primary Role:** The repository is responsible for interacting with the database, storing, and retrieving data. It abstracts the data access layer, providing a clean interface for services to interact with the data source.
2. **Components:**
 - **appsettings.json:** Contains configuration data, including database connection strings, environment settings, and other API configuration values.
 - **resourcecomponents.json, generatednames.json:** Files that store predefined resources or configuration data used by the API.
3. **Role in API Functionality:** The repository stores and retrieves data for the API. For example, when a request is made to fetch resource locations, the *ResourceLocationService.cs* will query the repository (e.g., *resourcecomponents.json*) to get the requested data.

Helpers

Location: *Helpers* Directory

1. **Primary Role:** Helper classes provide utility functions that support the operations of the controllers, services, and models. These can include validation, logging, data transformation, etc.
2. **Components:**
 - **LogHelper.cs:** Provides functionality for logging various events and errors across the API.
 - **ValidationHelper.cs:** Contains logic for validating incoming data, ensuring that requests meet necessary criteria before being processed.
3. **Role in API Functionality:** Helpers are used by services or controllers to avoid duplication of code. For example, *ValidationHelper.cs* would be used by a service to validate a new policy before it's saved to the database.

Configuration and Settings

Location: *appsettings.json* and *programsettings.json*

1. **Primary Role:** Configuration files store settings related to the operation of the API, such as database connections, authentication mechanisms, and API behavior.
2. **Components:**
 - **appsettings.json:** Contains environment-specific configuration settings like connection strings and logging configurations.
 - **programsettings.json:** Stores program-specific settings like user permissions or specific feature toggles.
3. **Role in API Functionality:** These configuration files ensure the API runs with the correct parameters depending on the environment (e.g., development, production). They can store sensitive data or configuration that the API needs to function properly, such as API keys or server endpoints.

Security and Authentication

Location: Various files (e.g., *Startup.cs*, middleware)

1. **Primary Role:** Security components ensure that only authorized users can interact with the API, especially for sensitive operations.
2. **Components:**
 - **Authentication middleware:** Ensures that incoming requests include valid authentication tokens (e.g., JWT).
 - **Role-based authorization:** Ensures that users can only access routes they are authorized to use, based on their roles (e.g., admin vs. user).
3. **Role in API Functionality:** The authentication and authorization logic ensures secure access control. For example, before a user can create a new resource or modify data, the API checks if the user has the correct permissions (typically handled in middleware).

Static Files and Assets

Location: *wwwroot* Directory

1. **Primary Role:** Static files like images, CSS, and JavaScript are served directly to the client without being processed by the backend.
2. **Components:**
 - **images/:** Contains logos, icons, and other media files used by the frontend.
 - **css/:** Holds stylesheets, including third-party libraries like Bootstrap.
 - **js/:** Contains JavaScript libraries used for client-side interactivity.
3. **Role in API Functionality:** While static assets don't process data, they provide essential frontend functionality. The API might serve these files to the client, but it doesn't directly handle their logic. For example, the API might return a 404 error page that is styled using *site.css*.

Technical Requirements:

5. Configuration and Setup

Deployment Options

1. **Run as an Azure Web App Using GitHub Action**
 - **Description:** App Service running a .NET application.
 - **Ideal for:** Fastest deployment.
 - **Requirements:**
 - An Azure Web App.
 - Utilizes the provided GitHub Action for deployment.
 - Requires GitHub secrets
2. **Run as a Docker Image**
3. **Run as a Docker Image with Podman**
4. **Run as a Web App for Containers**
5. **Run as an Azure Container App**
6. **Run as a Stand-Alone Site**

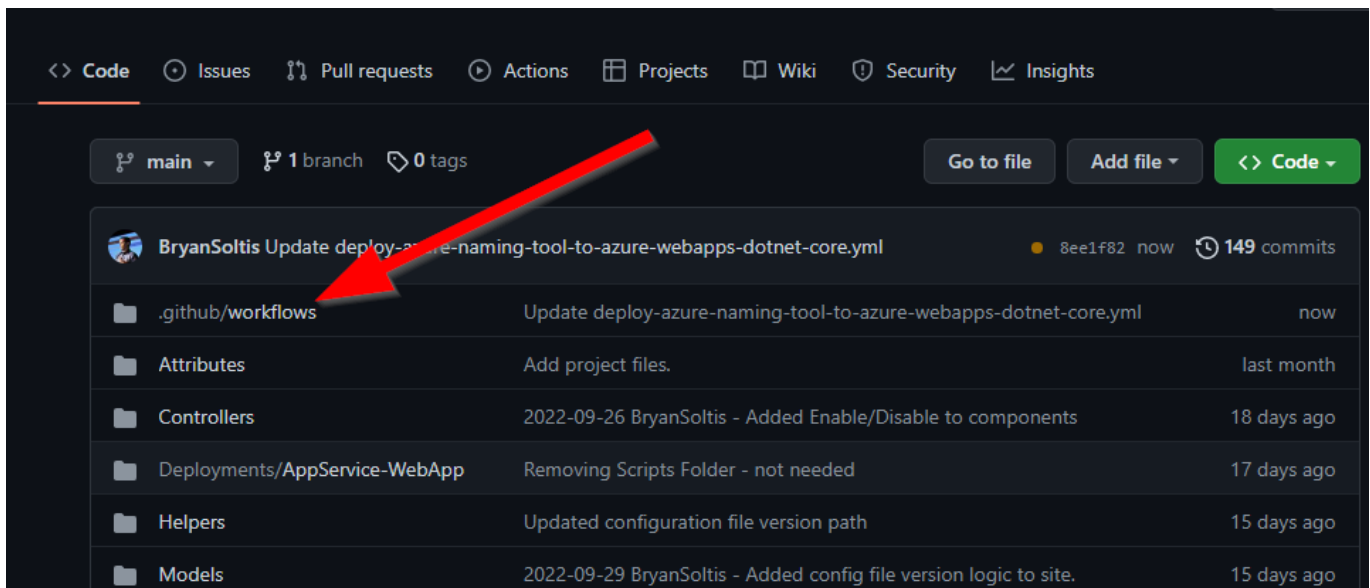
Installation Process as Azure Web App Using GitHub Action

This process allows you to deploy the Azure Naming Tool as a .NET 8 application in Azure Web App. This is the fastest deployment option, enabling you to utilize your installation in minutes. The provided GitHub Action automatically deploys your repository code on every commit, providing a secure and scalable solution.

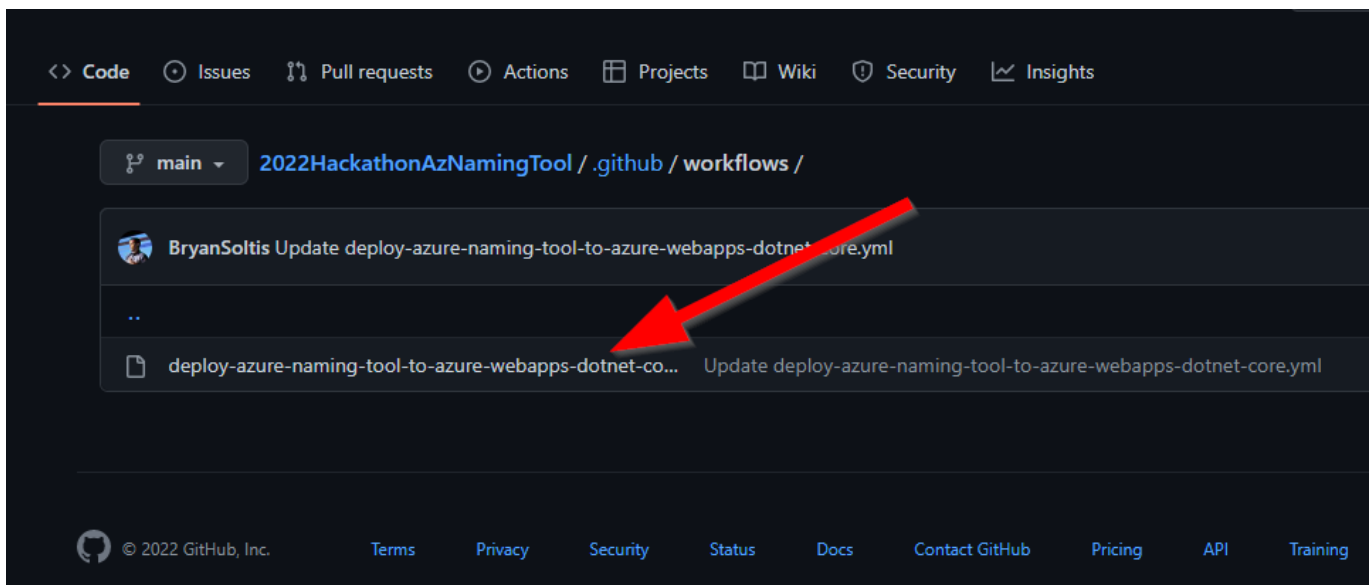
High-Level Installation Steps

1. Fork the Azure Naming Tool Repository

- Click on the [AzureNamingTool](#) link to open the repository.
- Click the **Fork** option in the top-right menu.
- Select your desired Owner and Repository name, then click **Create fork**.
- Click the green **<> Code** button.
- Navigate to `.github/workflows`.



- Click on `deploy-azure-naming-tool-to-azure-webapps-dotnet-core.yml` to view instructions for creating the required GitHub secrets.



NOTES:

- The GitHub Action will not deploy successfully until the secrets are created.
- You must create an Azure Web App and configure the GitHub Action secrets to deploy to your Azure Web App.

2. Create an Azure Web App

- Create a new Azure Web App in the Azure portal.
- **Leave everything as Default except the following**
- For the Publish option, select **Code**.
- For the Runtime stack, select **.NET 8**.

Microsoft Azure

Home > Create a resource >

Create Web App

Basics Deployment Networking Monitor + secure Tags Review + create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#)

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Resource Group * [Create new](#)

Instance Details

Name ☒ Try a secure unique default hostname. [More about this update](#)

Publish * ☒ Code ☐ Container

Runtime stack *


Operating System * ☒ Linux ☐ Windows

Region *

Pricing plans

[Review + create](#) [< Previous](#) [Next : Deployment >](#)

Create Web App ...

 Configuring deployment with GitHub Actions during app creation isn't supported with your selections of operating system and App Service plan. If you want to keep these selections, you can configure deployment with GitHub Actions after the web app is created.

GitHub settings

Set up GitHub Actions to push content to your app whenever there are code changes made to your repository. Note: Your GitHub account must have write access to the selected repository in order to add a workflow file which manages deployments to your app.

GitHub account

KXS-UTD

Change account



Organization

Select organization



Repository

Select repository




Branch

Select branch



Workflow configuration

Click the button below to preview what the GitHub Actions workflow file will look like before setting up continuous deployment.

 Complete the Basics tab and the form above to preview the GitHub Actions workflow file.

Preview file

Authentication settings

Choose if you would like to allow basic authentication to deploy code to your app. [Learn more](#)

Basic authentication



Disable



Enable

Review + create

< Previous





Next : Networking >

Create Web App ...

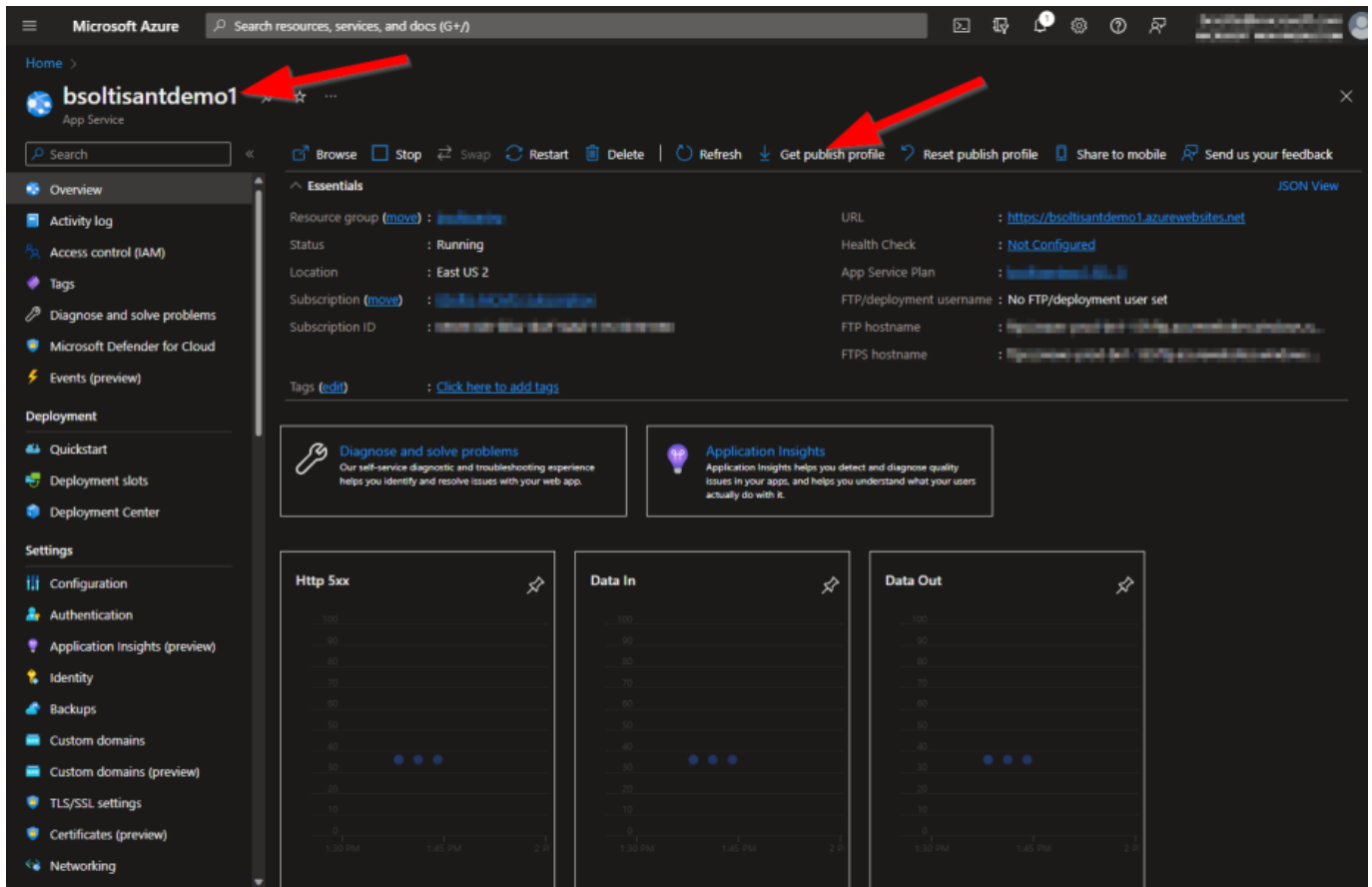
Basics Deployment Networking Monitor + secure **Tags** Review + create

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.

Note that if you create tags and then change resource settings on other tabs, your tags will be automatically updated.

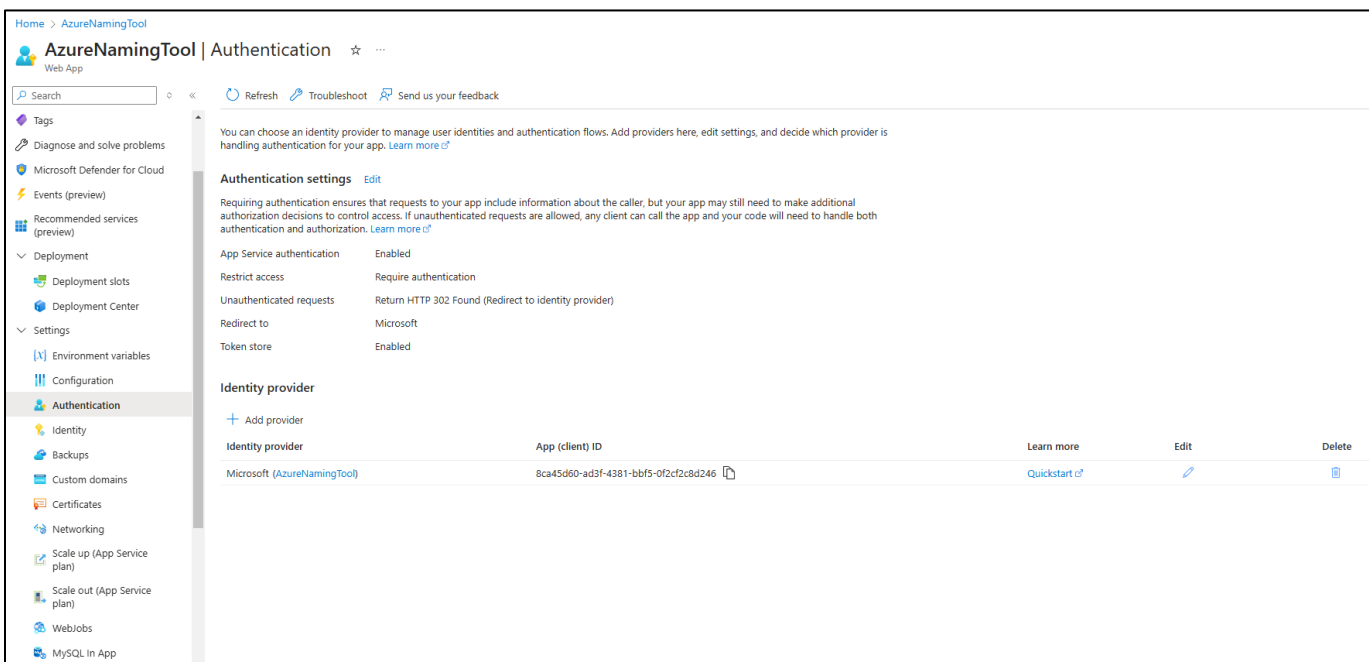
Name ⓘ	Value ⓘ	Resource	
<input type="text" value="CostCenter"/>	: <input type="text" value="Capstone"/>	3 selected	
Name	: UTD ECDF	All resources	
ms-resource-usage	: azure-cloud-shell	All resources	
<input type="text"/>	: <input type="text"/>	3 selected	

- Download the Publish Profile for use within the GitHub Action secret:
- `Get-AzWebApp -Name <webappname> | Get-AzWebAppPublishingProfile -OutputFile <filename> | Out-Null`
- **OR** retrieve the Publish Profile from the Azure portal.



3. Enable Azure Web App Authentication

- In the Azure Portal for your Azure Web App, navigate to the **Authentication** blade.
- Select **Add identity provider**.
- In the Identity provider section, select **Microsoft**.
- Enter a desired Name (default options can be left unchanged).
- Click **Add**.



4. Create GitHub Secrets

- In your GitHub repository, click **Settings** in the top menu.
- Click **Secrets** in the left menu.
- Click **New repository secret**.
 - Enter `AZURE_WEBAPP_PUBLISH_PROFILE` as the Name.
 - Enter the Publish Profile data for your Azure Web App as the Value.
- Click **Add secret**.
- Click **New repository secret** again.
 - Enter `AZURE_WEBAPP_NAME` as the Name.
 - Enter the name of your Azure Web App as the Value.

The screenshot shows the GitHub repository settings page. On the left sidebar, the 'Secrets and variables' section is expanded, with 'Actions' selected. The main content area is divided into two tabs: 'Secrets' and 'Variables'. The 'Secrets' tab is active, showing 'Environment secrets' and a message 'This environment has no secrets.' with a 'Manage environment secrets' button. Below this, the 'Repository secrets' section is visible, showing a list of secrets. The list has columns for 'Name' and 'Last updated'. The secrets listed are:

Name	Last updated
AZUREAPPSERVICE_CLIENTID_2F7B2CC7591047E986EE71866A968262	last month
AZUREAPPSERVICE_PUBLISHPROFILE_62CD84B9E45942D1978014E496145D8D	last month
AZUREAPPSERVICE_SUBSCRIPTIONID_886564E937F54ADAA3DCC397CE34F79	last month
AZUREAPPSERVICE_TENANTID_3234134C6C4C43AB980F3192440C87F6	last month
AZURE_WEBAPP_NAME	last month
AZURE_WEBAPP_PUBLISH_PROFILE	3 weeks ago

5. Enable GitHub Action

- In your GitHub repository, click the **Actions** tab.
- Click on **I understand my workflows, go ahead and enable them**.
- Select the **Azure Naming Tool - Build and deploy to an Azure Web App** workflow in the left navigation.
- Click **Run workflow** and confirm that the workflow completes successfully.

Actions New workflow

All workflows

- Azure Naming Tool - Build and deploy ...
- Build and deploy ASP.Net Core app to ...
- Build and deploy ASP.Net Core app to ...
- CodeQL
- Docker

Management

- Caches
- Deployments
- Attestations
- Runners

Help us improve GitHub Actions

Tell us how to make GitHub Actions work better for you with three quick questions. Give feedback

8 workflow runs

This workflow has a workflow_dispatch event trigger. Run workflow

Update azure-pipelines.yml for Azure Pipelines	main	3 weeks ago 2m 28s	...
Update azure-pipelines.yml for Azure Pipelines	main	3 weeks ago 2m 16s	...
Update azure-pipelines.yml for Azure Pipelines	main	3 weeks ago 2m 23s	...
Update azure-pipelines.yml for Azure Pipelines	main	3 weeks ago 4m 49s	...
Update azure-pipelines.yml for Azure Pipelines	main	3 weeks ago 18m 16s	...
Azure Naming Tool - Build and deploy to an Azure Web App	main	last month 5m 8s	...
Add or update the Azure App Service build and deployment workflow config	main	last month 13s	...
Add or update the Azure App Service build and deployment workflow config	main	last month 3h 28m 43s	...

6. Access Your Azure Web App

- Confirm that the site is successfully deployed.
- NOTE:** If your website does not load, check the configuration to ensure the dotnet AzureNamingTool.dll command is set on startup.

Search

Refresh Save Discard Leave Feedback

Custom Error pages requires a premium App Service Plan.

Application settings General settings Path mappings Error pages (preview)

Stack settings

Stack .NET

Major version .NET 8 (LTS)

Minor version .NET 8 (LTS)

Startup Command dotnet AzureNamingTool.dll

Provide an optional startup command that will be run as part of container startup. [Learn more](#)

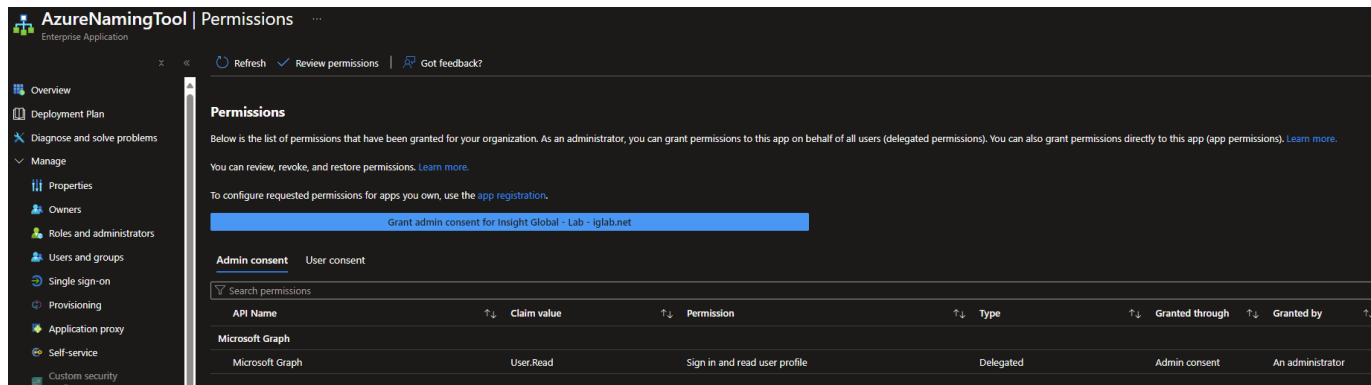
Platform settings

SCM Basic Auth Publishi... ☒ On ☐ Off

FTP Basic Auth Publishi... ☒ On ☐ Off

Additional Permissions

Make sure to grant **User.Read** permission in the consent for Microsoft Graph to enable proper user authentication.



Naming Policy Configuration

This guide explains how to configure the Azure Naming Tool to align with Insight Global's naming policy. Follow these steps to customize the naming structure based on your organization's requirements.

Prerequisites

- Obtain the naming policy document (in PDF format) from the development team. This document outlines the custom naming conventions and rules for various Azure resources.
- (Optional) Obtain the tagging policy document, if required, for reference. Note: The Azure Naming Tool focuses only on naming; tagging may need to be implemented separately, potentially in the deployment pipeline.

Steps to Configure the Naming Tool

- Access the Configuration Page
 - Navigate to the configuration page of the Azure Naming Tool. This is where you will define and customize naming rules.

Azure Naming Tool

User: GlobalAdmin

Configuration

This page contains the inputs to customize the Azure Naming Tool configuration.

▼ Actions Legend

Base Configuration

▼ Components

▼ Delimiters

Component Configuration

▼ Environments

▼ Functions

▼ Locations

▼ Orgs

▼ Projects/Apps/Services

▼ Resource Types

▼ Units/Depts

General Configuration

▼ Global Configuration

2. Set the Components Section

- Locate the Components Section.
- Define the order of components as specified in the naming policy.
 - For example: Tier → Region → Resource Type → Workload/App/Service → Instance Number.
- This order determines how the names of Azure resources will be constructed.























Components	
Current Components	
The current Components.	
+ Add Component	
Name	Actions
Resource Types	↑ ↓ ✎ ⊗
Orgs	↑ ↓ ✎ ⊗
Units/Depts	↑ ↓ ✎ ⊗
Projects/Apps/Services	↑ ↓ ✎ ⊗
Functions	↑ ↓ ✎ ⊗
Environments	↑ ↓ ✎ ⊗
Locations	↑ ↓ ✎ ⊗
Instance	↑ ↓ ✎ ⊗

3. Configure the Delimiter

- Go to the Delimiter Section.
- Specify the delimiter that separates each component in the resource name.
 - By default, the delimiter is set to -. Update this if your organization uses a different delimiter.

4. Customize Component Configurations

- Under the Component Config Section, set the specific names and values for each component:
 - Resource Types: Define custom names for resource types (e.g., VMs, Storage Accounts, etc.).
 - Applications/Workloads: Add or customize application names or workload identifiers.
 - Locations: Include Azure regions and any custom location identifiers used by the organization.
 - Environments: Specify environment names (e.g., DEV, TEST, PROD).
- Ensure these align with the entries provided in the naming policy document.

Name	Short Name	Optional	Excluded	Actions
AnalysisServices/servers	as	Units/Depts	Orgs Functions	 
ApiManagement/service	apim	Units/Depts	Orgs Functions	 
ApiManagement/service/api-version-sets	apivs	Units/Depts	Orgs Functions	 
ApiManagement/service/apis	apis	Units/Depts	Orgs Functions	 
ApiManagement/service/apis/issues	apii	Units/Depts	Orgs Functions	 
ApiManagement/service/apis/issues/attachments	apiia	Units/Depts	Orgs Functions	 
ApiManagement/service/apis/issues/comments	apiic	Units/Depts	Orgs Functions	 
ApiManagement/service/apis/operations	apio	Units/Depts	Orgs Functions	 
ApiManagement/service/apis/operations/tags	apiot	Units/Depts	Orgs Functions	 
ApiManagement/service/apis/releases	apir	Units/Depts	Orgs Functions	 
ApiManagement/service/apis/schemas	apis	Units/Depts	Orgs Functions	 

5. Adjust Global Configuration

- In the Global Configuration Section, you can:
- Import a predefined configuration file (if available) to apply an existing setup.
- Export the current configuration for sharing or backup purposes.

6. Verify and Save Changes

- Review the configured settings to ensure they match the naming policy.
- There's no need to save the changes as the changes are saved in real-time.

Notes

- The tool currently supports naming configurations only. If tagging is also required, consider implementing the tagging policy directly in the deployment pipeline.
- At the time of writing, the default naming configuration order is:
 - Tier → Region → Resource Type → Workload/App/Service → Instance Number

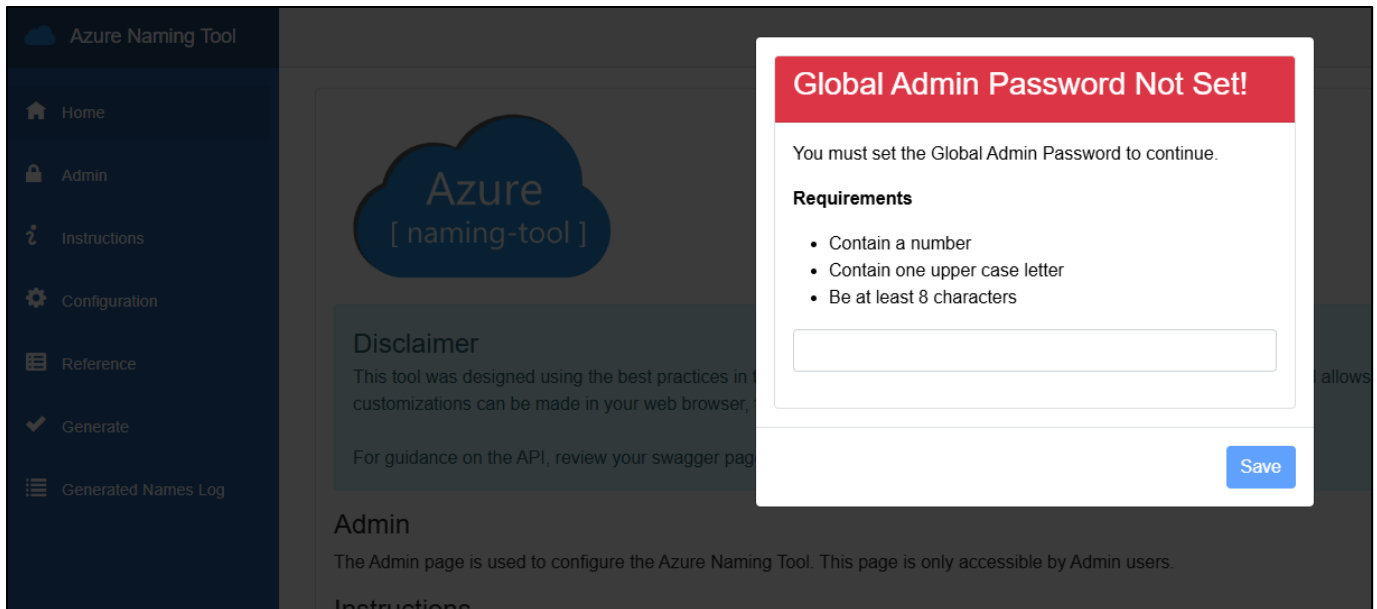
6. Usage and Maintenance

First Time Use

This guide will walk you through the initial setup and provide an overview of all the available pages in the tool.

Launch the Tool

- Open your browser and navigate to the Azure Naming Tool's URL.
- Upon first launch, you will be prompted with a screen asking for an **admin password**. Set the password and store it securely. This password will be required to access admin features.



Admin Page Overview

The **Admin Page** allows administrators to manage various configurations, security settings, and customizations for the Azure Naming Tool. Here's a detailed breakdown of the features available on this page:

Customization Section

This section enables you to customize the appearance and navigation of the Azure Naming Tool:

- **Custom Home Content:** You can enter **Markdown** content to be displayed on the home page. This can be used for providing additional information, announcements, or guidelines to users.
- **Custom Logo:** Upload a **custom logo** for the site.
 - **Note:** If the tool is hosted in a container, the uploaded logo will not persist if the container is recreated. You'll need to re-upload the logo if this happens.
 - **Site Navigation:** Configure which navigation links are available for **non-admin users**. You can enable or disable links to the following pages: Instructions Page, Configuration Page, Reference Page and Generated Names Log

Security Section

This section allows you to manage the **Global Admin password** and **API keys**:

- **Global Admin Password:** You can update the Global Admin password for the site.
- **API Keys:**

Two types of API keys are available:

 - **Full Access API Key:** Grants full access to the tool's API.
 - **Read-Only API Key:** Provides read-only access to the API (only supports GET operations).
 - You can **generate new API keys** or update the existing ones by clicking on the "Generate" button or manually updating the key.

▼ API Keys

▼ Full Access API Key

The current **Full Access API Key** is displayed. This key provides **full access** to the API.

Click **Generate** to create a new random **Full Access API Key**, or update the text to the desired value.

9c7cb02d-6634-4361-bb3f-91cf40844fbe

Save

Generate

▼ Read-Only API Key

The current **Read-Only API Key** is displayed. This key provides **read-only** access to the API (GET operations only).

Click **Generate** to create a new random **Read-Only API Key**, or update the text to the desired value.

886aa8d7-ab8c-44c3-babb-535a1fbab929

Save

Generate

Identity Provider Settings

This section allows you to configure the identity provider settings for authentication:

- **Identity Header Name:** This setting defines the header used to authenticate users via an Identity Provider (e.g., Azure App Service Authentication). The default is X-MS-CLIENT-PRINCIPAL-NAME.
- **Admin Users:** To assign users as admins, enter their user IDs in the field provided. After the change, users will need to **refresh their browser** to see the updated permissions.

Cache Section

The **Cache** section helps you manage cached data for the Azure Naming Tool:

- **View Cache:** Click **View** to inspect the cached data stored by the tool.
- **Clear Cache:** Click **Clear** to delete all cached data. This may help resolve issues caused by outdated information.

Site Settings Section

This section provides options to configure general site behavior:

- **Allow Duplicate Names:** By default, the tool prevents the generation of duplicate resource names. Enable this setting if you want to allow duplicate names.
- **Auto-Increment Resource Instance:** When enabled, this setting will automatically increment the resource instance number for each generated resource type name (e.g., app01 → app02).
- **Resource Type Editing:** This setting allows you to override the default **Azure portal resource type validation**, enabling customization of resource types if needed.
- **Connectivity Check:** This verifies the tool's ability to connect to the internet for updates. You can disable this check if necessary.

- **Generation Webhook:** You can configure a **webhook URL** to send generated names to another system. This is useful for integrating with external tools or processes.
- **Reset Site Settings:** This will **reset all site settings** to their default values, including customizations and configurations. **This action cannot be undone.**

Version Details

In this section, you'll find information about the current version of the Azure Naming Tool, including the version number and any update history.

Admin Log


The **Admin Log** page displays a log of administrative actions and configuration changes within the Azure Naming Tool, allowing administrators to track system events and issues.

Admin Log


This page displays a log of Admin/Configuration changes.


Export Admin LogPurge Admin Log

▼ Filter Data

10/21/2024 

to

11/20/2024 



Filter by Source, Title, Message

Reset

Created On	Source	Title	Message
11/20/2024 18:26:21	System	ERROR	A task was canceled.
11/20/2024 18:24:33	System	ERROR	The key {a11f72bb-c820-45ce-86dd-ed50

Filtering Data

The **Admin Log** page provides filtering options to help administrators narrow down log entries:

- **Date Range:** Filter logs based on a specific date range by selecting the **From** and **To** date fields.
- **Custom:** Filter logs based on the source, title (e.g., **ERROR**, **INFO**), by specific keywords or text within the message field.

Log Entries

The logs are displayed with the following columns: Created On, Source, Title and Message

Export Button

The **Export** button allows administrators to download the log data in a file format for further analysis or record-keeping.

Purge Button

The **Purge** button enables administrators to clear all the logs displayed on the page. This action is **irreversible**.

Configuration Page

The Configuration page allows administrators to customize the Azure Naming Tool to meet specific organizational needs. It includes inputs for setting up naming conventions, components, delimiters, and other configurations.

Actions Legend

The **Actions Legend** section provides an overview of the different actions that can be performed on the configuration page. This serves as a guide to help administrators navigate and understand the various settings available.

Base Configuration

This section contains the foundational settings for the tool's configuration, including the structure for resource naming.

- **Components Configuration:** The components section allows you to customize the order of the various naming components. For example, the order might specify if the **Environment** name should come first or if **Resource Types** should be included before or after other components.
- **Delimiters:** Here, you can configure the delimiter that separates naming components. By default, this delimiter is set to "-", but it can be customized to meet your specific naming conventions.

Component Configuration

This section contains the individual settings for each of the naming components. Each component represents a part of the naming structure and can be customized as needed.

- **Environments:** Define the naming convention for environments (e.g., **Dev**, **Prod**, **Test**).
- **Functions:** Customize naming for functions, such as specifying function identifiers or related descriptors.
- **Locations:** Set up location-based naming rules, which might include geographic locations or data center identifiers.
- **Orgs:** Configure naming for organizational units, departments, or business units.
- **Projects/Apps/Services:** Customize naming for different projects, applications, or services within your organization.
- **Resource Types:** Define how resource types should be named, such as virtual machines, databases, or storage accounts.
- **Units/Depts:** Set up naming conventions for units or departments within your organization.

General Configuration

This section contains more global settings for the Azure Naming Tool.

- **Global Configuration:** Here, administrators can configure settings that apply across the entire tool, such as importing and exporting configurations or resetting to default settings.

Generate Page

The Generate page is used to generate Azure resource names based on the selected naming components and resource types. Administrators or users can customize the components and choose whether to generate a name for a single resource or multiple resources.

Generate

This page generates a name for the selected resource type.

Instructions

1. Select the desired Resource Type / Resource Types
2. Review the Naming Guidelines (for single resource type)
3. Select the component options
4. Click **Generate** to generate the Azure resource name(s)
5. View generated names in the [Generated Names Log](#)

Generate Mode

☒ **Generate Single Resource Type Name**

This option will allow the generation of a single resource type name using the selected component options.

☐ **Generate Multiple Resource Type Names**

This option will allow the generation of names for multiple resource types using the selected component options.

Resource Type

Select the desired resource type.



Automation



Automation/automationAccounts/certificates (cert)

Clear

This resource has a lower-level scope and does not require a unique name.

Instructions

This section provides a step-by-step guide to generating resource names.

- **Select the desired Resource Type(s):** Choose the resource type(s) you want to generate names for (e.g., virtual machines, storage accounts, etc.).
- **Review the Naming Guidelines (for a single resource type):** Refer to the guidelines for the selected resource type to ensure the generated names meet the required naming standards.

- **Select the component options:** Choose the components that should be included in the resource name, such as environment, region, or resource type.
- **Click Generate:** After configuring the options, click the **Generate** button to create the resource name(s).
- **View generated names in the Generated Names Log:** Once the names are generated, you can view them in the **Generated Names Log** page for record-keeping.

Generate Mode

The **Generate Mode** section provides two options for generating resource names:

- **Generate Single Resource Type Name:** This option allows you to generate a name for a **single resource type** using the selected components. You can customize each component, such as environment, region, and resource type, to fit your naming conventions.
- **Generate Multiple Resource Type Names:** This option allows you to generate names for **multiple resource types** at once. You will need to select all required components, including any optional or excluded components. This option is useful if you need to generate names for several resources simultaneously.

Resource Type

The **Resource Type** section allows you to:

- **Select the desired resource type:** Choose the specific resource type for which you want to generate a name (e.g., **Virtual Machine, Storage Account, App Service**).
- **Filter by Category:** You can filter resource types by category to make it easier to find the specific resource you're looking for.
- **Search Resource Types:** A search bar allows you to quickly find the desired resource type by typing in keywords or the full name of the resource type.

Generated Names Log

The Generated Names Log page displays a log of all the generated resource type names. It helps track the names that have been generated, including details about each name, the resource type, and the components used in the generation process.

Log Details

This section lists the generated names along with associated details such as: Created On, Created By, Generated Name, Resource Type, Components and Message

Filter Data

You can filter the generated names by the following criteria:

- **Date Range:** Filter the logs by a specific date range (e.g., from 10/21/2024 to 11/20/2024).
- **Custom:** Created By, Generated Name, Resource Type, Components.

Export and Purge

- **Export:** This option allows you to export the generated names log to an external file (e.g., CSV) for record-keeping or analysis.
- **Purge:** The **Purge** option is used to remove logs from the system. However, purging is restricted to prevent accidental loss of valuable data.

Next Steps After Initial Setup

- Once you've completed the initial setup, you can proceed to customize naming configurations based on the requirements outlined in your organization's policy documents.
- Explore each page and begin generating names for Azure resources as needed.

API Usage

The Azure Naming Tool's API allows programmatic access to its functionalities. Here's how to interact with it:

1. Access the Swagger Page

- Navigate to the Swagger interface for guidance on using the API.

The screenshot displays the Swagger UI for the Azure Naming Tool API. At the top, the Swagger logo is visible, along with the text 'Supported by SMARTBEAR' and a 'Select a definition' button. The main heading is 'Azure Naming Tool API', accompanied by version tags 'v4.2.1' and 'OAS 3.0'. Below this, the Swagger file path '/swagger/v1/swagger.json' is shown. A descriptive paragraph states: 'An ASP.NET Core Web API for managing the Azure Naming tool configuration. All API requests require the configured API Keys (found in the site Admin configuration [documentation](#)).' The 'Admin' section lists nine API endpoints, each with its HTTP method, path, and a brief description of its function.

Method	Path	Description
POST	/api/Admin/UpdatePassword	This function will update the Global Admin Password.
POST	/api/Admin/UpdateAPIKey	This function will update the Full Access API Key.
POST	/api/Admin/GenerateAPIKey	This function will generate a new Full Access API Key.
POST	/api/Admin/UpdateReadOnlyAPIKey	This function will update the Read-Only API Key.
POST	/api/Admin/GenerateReadOnlyAPIKey	This function will generate a new Read-Only API Key.
GET	/api/Admin/GetAdminLog	This function will return the admin log data.
POST	/api/Admin/PurgeAdminLog	This function will purge the admin log data.
GET	/api/Admin/GetGeneratedNamesLog	This function will return the generated names data.
GET	/api/Admin/GetGeneratedName/{id}	This function will return the generated names data by ID.

2. Sending a Resource Naming Request

ResourceNamingRequests

POST

/api/ResourceNamingRequests/RequestNameWithComponents
This function will generate a resource type name for specified component values. This function requires full definition for all components. It is reco

POST

/api/ResourceNamingRequests/RequestName This function will generate a resource type name for specified component values, us

POST

/api/ResourceNamingRequests/ValidateName
This function will validate the name for the specified resource type. NOTE: This function does not validate using the tool configuration, only the req
validate using the tool configuration.

ResourceOrgs

- For generating a resource name first add the API key (Admin key)
- Expand the naming request section in Swagger.
- Paste your API key and provide a JSON payload with necessary details (resource type, project, location, environment).

POST

/api/ResourceNamingRequests/RequestName This function will generate a resource type name for specified component valu

Parameters

Name	Description
APIKey * required string (header)	<div>9c7cb02d-6634-4361-bb3f-91cf40844fbe</div>

Request body

ResourceNameRequest (json) - Resource Name Request data

```
{  "resourceEnvironment": "dev",  "resourceFunction": "func",  "resourceInstance": "001",  "resourceLocation": "use",  "resourceOrg": "so",  "resourceProjAppSvc": "spa",  "resourceType": "as",  "resourceUnitDept": "sud",  "customComponents": {    "additionalProp1": "sample"  },  "resourceId": 0,  "createdBy": "sample"}
```

Execute

- Execute the request to receive a generated resource name in the response.

```
{
  "resourceId": 0,
  "createdBy": "sample"
}
```

Request URL

http://localhost:8081/api/ResourceNamingRequests/RequestName

Server response

Code	Details
------	---------

200

Response body

```
{
  "resourceName": "assudspadevuse001",
  "message": "This resource type only allows lowercase names. The generated name has been updated to lowercase characters of the name exceeding the max length and the delimiter removed to shorten the value or the delimiter is not an allowed character",
  "success": true,
  "resourceNameDetails": {
    "id": 1,
    "createdOn": "2024-11-20T18:36:52.9975059+00:00",
    "resourceName": "assudspadevuse001",
    "resourceTypeName": "AnalysisServices/servers",
    "components": [
      [
        "ResourceType",
        "as"
      ],
      [
        "ResourceUnitDept",
        "sud"
      ],
      [
        "ResourceProjAppSvc",
        "spa"
      ],
      [
        "ResourceEnvironment",
        "dev"
      ]
    ]
  }
}
```

Response headers

```
content-type: application/json; charset=utf-8
date: Wed, 20 Nov 2024 18:36:52 GMT
server: Kestrel
transfer-encoding: chunked
```

Code	Details
200	<div>Response body</div> <pre> "as"], ["ResourceUnitDept", "sud"], ["ResourceProjAppSvc", "spa"], ["ResourceEnvironment", "dev"], ["ResourceLocation", "use"], ["ResourceInstance", "001"]], "user": "API", "message": "This resource type only allows lowercase names. The generated name has been u gth of the name exceeding the max length and the delimiter removed to shorten the value or th } } </pre> <div>Response headers</div> <pre> content-type: application/json; charset=utf-8 date: Wed,20 Nov 2024 18:36:52 GMT server: Kestrel transfer-encoding: chunked </pre>

3. Executing a GET Request

- Use the Try it out button on the Swagger page to execute a GET request for resource delimiters. Paste your API key and execute the call.
- The response will show the current delimiter configuration (e.g., "dash").

4. Handling Errors

- If required fields are missing in your request, you will receive a 400 response with an error message indicating what is required.

Update Strategy

- Refer to GitHub source repo for latest updates. [AzureNamingTool-Releases](#)
- Backup all configuration JSON files before proceeding with any updates. **For backup, go to configuration, scroll down to global configuration and export the current global configuration.**
- Follow the specific instructions based on your deployment method (Docker, Azure Web App, .NET site).

Deep Dive Resources

- For a deeper understanding of the tool's architecture and configuration processes, refer to [Bryan Soltis' Deep Dive Blog](#).