KodeKloud



Visit www.kodekloud.com to learn more.

# Managing Container Images in Azure Container Registry

# Introduction

01 Understand the features offered by Azure Container Registry and its benefits

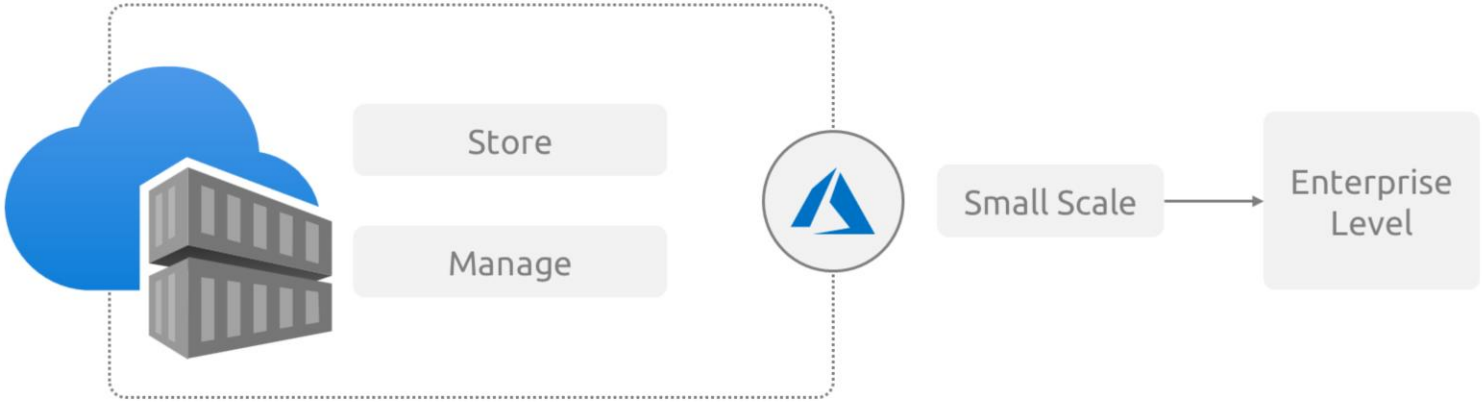
02 Learn how to automate builds using ACR Tasks

03 Describe the elements in a Dockerfile

04 Learn to build and push image in an ACR using Azure CLI

# Discovering Azure Container Registry

# Discovering the Azure Container Registry

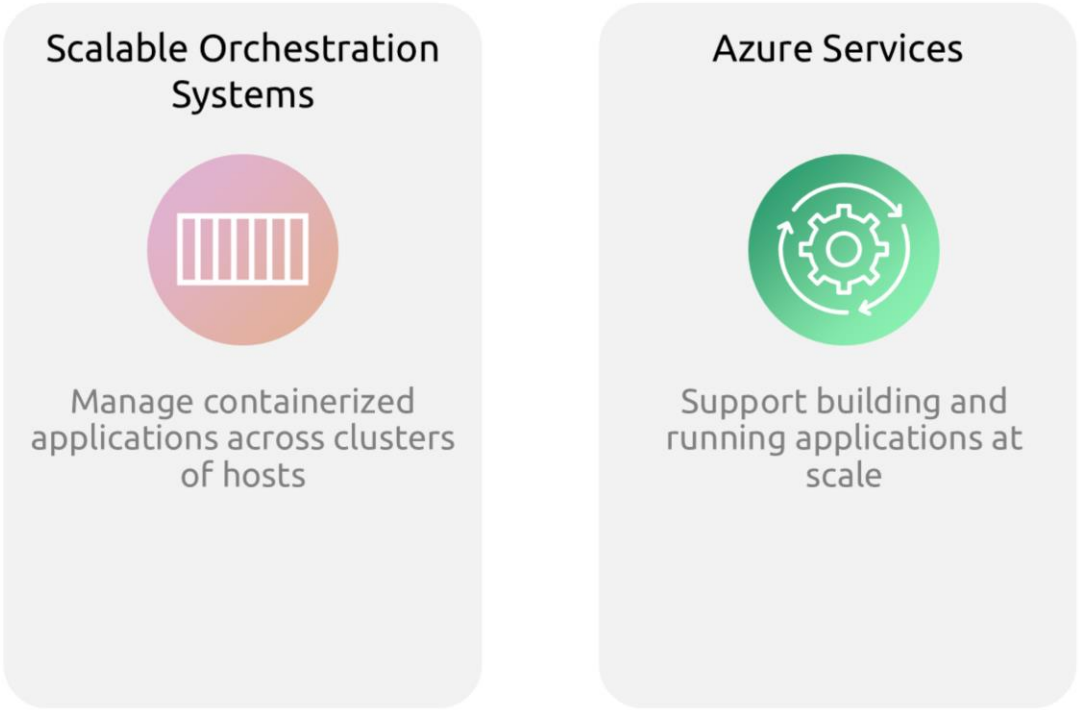





Azure Container Registry, or a CR is a managed docker registry that allows you to store and manage container images for deployment in Azure and other environments It supports scenarios from small scale containerized applications to enterprise level Solutions Let's explore some of the key use cases where a CR can help your applications run smoothly

# Discovering the Azure Container Registry

Use cases





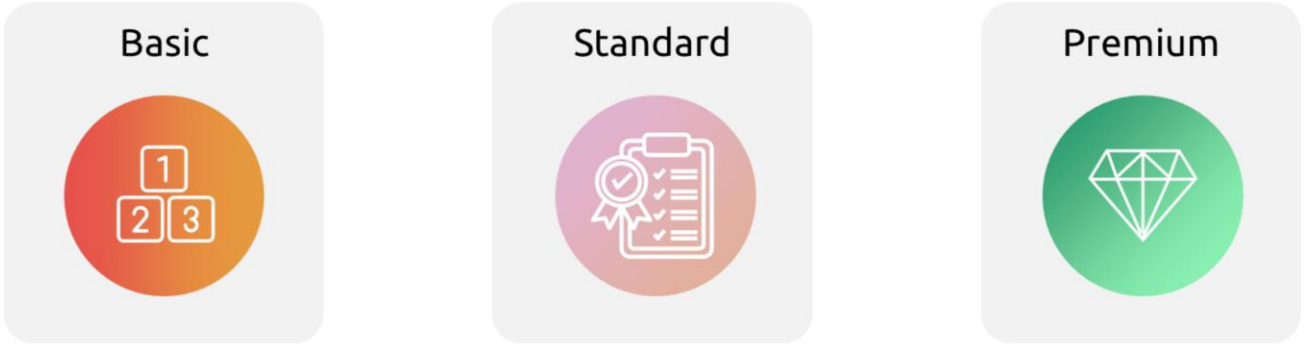

Use the Azure Container Registry (ACR) service with your existing container development and deployment pipelines or use Azure Container Registry Tasks to build container images in Azure.

Use cases
Scalable orchestration systems: including Kubernetes, DC/OS, and Docker Swarm.
Azure services: including Azure Kubernetes Service (AKS), App Service, Batch, Service Fabric, and others.
Developers can also push to a container registry as part of a container development workflow. For example, target a

container registry from a continuous integration and delivery tool such as Azure Pipelines or Jenkins. Configure ACR Tasks to automatically rebuild application images when their base images are updated, or automate image builds when your team commits code to a Git repository. Create multi-step tasks to automate building, testing, and patching multiple container images in parallel in the cloud.

# Discovering the Azure Container Registry

## Azure Container Registry service tiers






Azure Container Registry service tiers
Azure Container Registry is available in multiple service tiers. These tiers provide predictable pricing and several options for aligning to the capacity and usage patterns of your private Docker registry in Azure.
Basic - A cost-optimized entry point for developers learning about Azure Container Registry. Basic registries have the same programmatic capabilities as Standard and Premium (such as Azure Active Directory authentication integration, image deletion, and webhooks). However, the included storage and image throughput are most appropriate for lower usage scenarios.

Standard - Standard registries offer the same capabilities as Basic, with increased included storage and image throughput. Standard registries should satisfy the needs of most production scenarios.
Premium - Premium registries provide the highest amount of included storage and concurrent operations, enabling high-volume scenarios. In addition to higher image throughput, Premium adds features such as geo-replication for managing a single registry across multiple regions, content trust for image tag signing, private link with private endpoints to restrict access to the registry.

# Discovering the Azure Container Registry

Supported images and artifacts

Each image is a read-only snapshot of a Docker-compatible container

Can include both Windows and Linux images

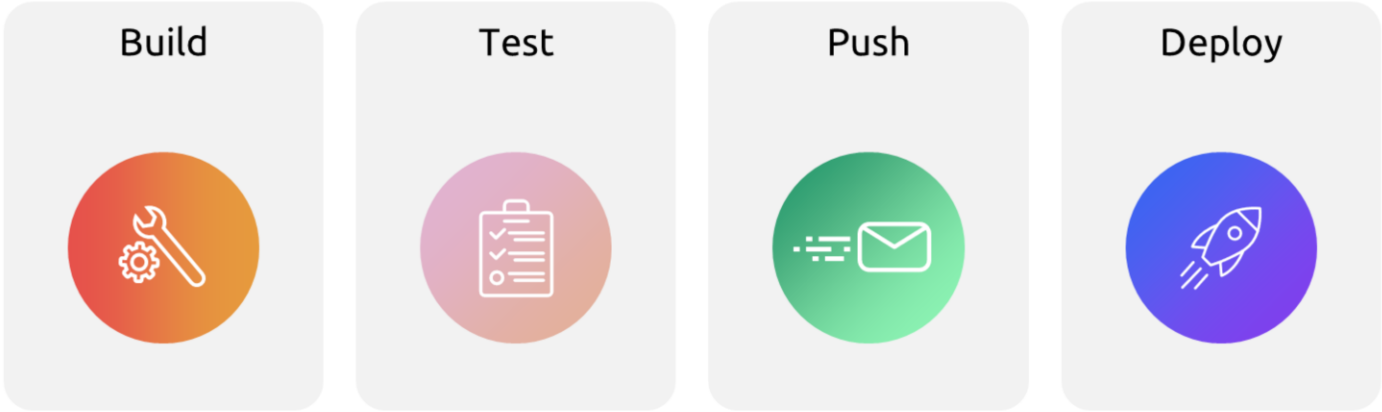
Stores Helm charts and images built to the Open Container Initiative (OCI) Image Format specification



ACR Tasks: Configure build tasks to automate your container OS and framework patching pipeline and build images automatically when your team commits code to source control.

# Discovering the Azure Container Registry

## Azure Container Registry Tasks






**ACR Tasks:** Configure build tasks to automate your container OS and framework patching pipeline and build images automatically when your team commits code to source control.

Use Azure Container Registry Tasks (ACR Tasks) to streamline building, testing, pushing, and deploying images in Azure.

# Storage Capabilities

### Encryption at Rest

All container images in your registry are encrypted at rest.

### Geo-Redundant Storage

Azure uses a geo-redundant storage scheme.

### Geo-Replication

Consider using the geo-replication feature of Premium registries.



Very high numbers of repositories and tags can impact the performance of your registry. Periodically delete unused repositories, tags, and images as part of your registry maintenance routine. Deleted registry resources like repositories, images, and tags cannot be recovered after deletion.

Every Basic, Standard, and Premium Azure container registry benefits from advanced Azure storage features.
Encryption-at-rest: All container images in your registry are encrypted at rest.
Geo-redundant storage: Azure uses a geo-redundant storage scheme to guard against loss of your container images.

Geo-replication: For scenarios requiring even more high-availability assurance, consider using the geo-replication feature of Premium registries.

# Exploring Elements of a Dockerfile

# Elements of a Dockerfile

```
# Step 1: Specify the parent image for the new image
FROM ubuntu:18.04
# Step 2: Update OS packages and install additional software
RUN apt -y update && apt install -y wget nginx software-properties-common apt-transport-https \
         && wget -q <URL>/ubuntu/18.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb \
         && dpkg -i packages-microsoft-prod.deb \
         && add-apt-repository universe \
         && apt -y update \
         && apt install -y dotnet-sdk-3.0
# Step 3: Configure Nginx environment
CMD service nginx start
# Step 4: Configure Nginx environment
COPY ./default /etc/nginx/sites-available/default
```



If you want to create a custom container you will need to understand the elements of a Dockerfile. A Dockerfile is a text file that contains the instructions we use to build and run a Docker image.

The following aspects of the image are defined:
The base or parent image we use to create the new image
Commands to update the base OS and install additional software
Build artifacts to include, such as a developed application

Services to expose, such a storage and network configuration
Command to run when the container is launched

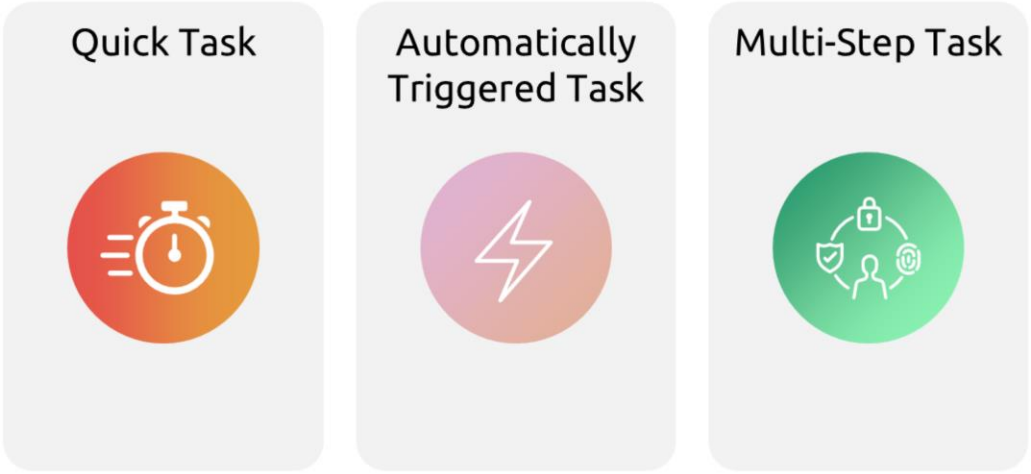
# Elements of a Dockerfile

```
# Step 5: Configure work directory
WORKDIR /app
# Step 6: Copy website code to container
COPY ./website/. .
# Step 7: Configure network requirements
EXPOSE 80:8080
# Step 8: Define the entry point of the process that runs in the container
ENTRYPOINT ["dotnet", "website.dll"]
```

# Building and Managing Containers With ACR Tasks

# Building and Managing Containers With Tasks

Task scenarios






ACR Tasks provides cloud-based container image building for platforms including Linux, Windows, and ARM. It can automate OS and framework patching for your Docker containers.

ACR Tasks supports several scenarios to build and maintain container images and other artifacts:

Quick task
Build and push a single container image to a container registry on-demand, in Azure, without needing a local Docker Engine

installation.
Before you commit your first line of code, ACR Tasks's quick task feature can provide an integrated development experience by offloading your container image builds to Azure. With quick tasks, you can verify your automated build definitions and catch potential problems prior to committing your code.

Trigger task on source code update
Trigger a container image build or multi-step task when code is committed, or a pull request is made or updated, to a public or private Git repository in GitHub or Azure DevOps. For example, configure a build task with the Azure CLI command az acr task create by specifying a Git repository and optionally a branch and Dockerfile. When your team updates code in the repository, an ACR Tasks-created webhook triggers a build of the container image defined in the repo.

Trigger on base image update
You can set up an ACR task to track a dependency on a base image when it builds an application image. When the updated base image is pushed to your registry, or a base image is updated in a public repo such as in Docker Hub, ACR Tasks can automatically build any application images based on it.

Schedule a task
Optionally schedule a task by setting up one or more timer triggers when you create or update the task. Scheduling a task is useful for running container workloads on a defined schedule, or running maintenance operations or tests on images pushed regularly to your registry.

Multi-step tasks
Extend the single image build-and-push capability of ACR Tasks with multi-step, multi-container-based workflows.
Multi-step tasks, defined in a YAML file specify individual build and push operations for container images or other artifacts. They can also define the execution of one or more containers, with each step using the container as its execution environment. For example, you can create a multi-step task that automates the following:

Build a web application image
Run the web application container
Build a web application test image
Run the web application test container, which performs tests against the running application container
If the tests pass, build a Helm chart archive package
Perform a helm upgrade using the new Helm chart archive package

# Running Container Images in Azure Container Instances

# Introduction

01 Explore Azure Container Instances and their benefits

02 Understand the ease of running containers using Azure Container Instances

# Exploring Azure Container Instances

# Azure Container Instances

| Feature | Description |
| --- | --- |
| Fast startup times | Containers can start in seconds, without the need to provision and manage VMs |
| Public IP connectivity and DNS name | Containers can be directly exposed to the internet with an IP address and a fully qualified domain name (FQDN) |
| Hypervisor-level security | Container applications are as isolated in a container as they would be in a VM |
| Custom sizes | ACI provides optimum utilization by allowing exact specifications of CPU cores and memory |
| Persistent storage | Containers support direct mounting of Azure Files shares |
| Linux and Windows containers | The same API is used to schedule both Linux and Windows containers |
| Co-scheduled groups | Container Instances supports scheduling of multicontainer groups that share host machine resources |
| Virtual network deployment | Container Instances can be deployed to an Azure virtual network |

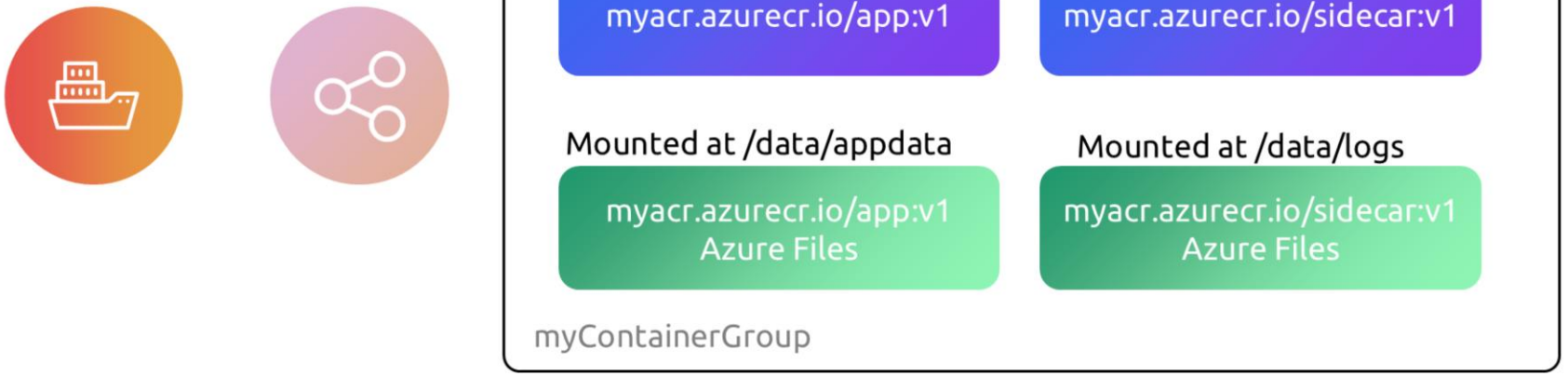


- Fast startup: ACI can start containers in Azure in seconds, without the need to provision and manage VMs
- Container access: ACI enables exposing your container groups directly to the internet with an IP address and a fully qualified domain name (FQDN)
- Hypervisor-level security: Isolate your application as completely as it would be in a VM
- Customer data: The ACI service stores the minimum customer data required to ensure your container groups are running as expected
- Custom sizes: ACI provides optimum utilization by allowing exact specifications of CPU cores and memory

- Persistent storage: Mount Azure Files shares directly to a container to retrieve and persist state
- Linux and Windows: Schedule both Windows and Linux containers using the same API.

For scenarios where you need full container orchestration, including service discovery across multiple containers, automatic scaling, and coordinated application upgrades, we recommend Azure Kubernetes Service (AKS).

# Azure Container Instances – Container Groups






Container groups
-The top-level resource in Azure Container Instances is the container group.
-The containers in a container group share a lifecycle, resources, local network, and storage volumes.

This example container group:
Is scheduled on a single host machine.
Is assigned a DNS name label.

Exposes a single public IP address, with one exposed port.
Consists of two containers. One container listens on port 80, while the other listens on port 5000.
Includes two Azure file shares as volume mounts, and each container mounts one of the shares locally.

Note: Multi-container groups currently support only Linux containers. For Windows containers, Azure Container Instances only supports deployment of a single instance.

# Azure Container Instances

## Deployment

A multi-container group can be deployed in two ways: ARM template or a YAML file.

## Resource Allocation

ACI allocates resources to a container group by adding the resource requests of the instances in the group.

## Networking

Container groups share an IP address and a port namespace on that IP address.

## Storage

- Specify external volumes to mount within a container group.
- Map those volumes into specific paths within the individual containers in a group.

## Common Scenarios

Multi-container groups are useful in cases where you want to divide a single functional task into a small number of container images.



Deployment
A Resource Manager template is recommended when you need to deploy additional Azure service resources (for example, an Azure Files share) when you deploy the container instances.
Due to the YAML format's more concise nature, a YAML file is recommended when your deployment includes only container instances.

Resource allocation

Taking CPU resources as an example, if you create a container group with two instances, each requesting 1 CPU, then the container group is allocated 2 CPUs.

Networking
To enable external clients to reach a container within the group, you must expose the port on the IP address and from the container. Because containers within the group share a port namespace, port mapping isn't supported. Containers within a group can reach each other via localhost on the ports that they have exposed, even if those ports aren't exposed externally on the group's IP address.

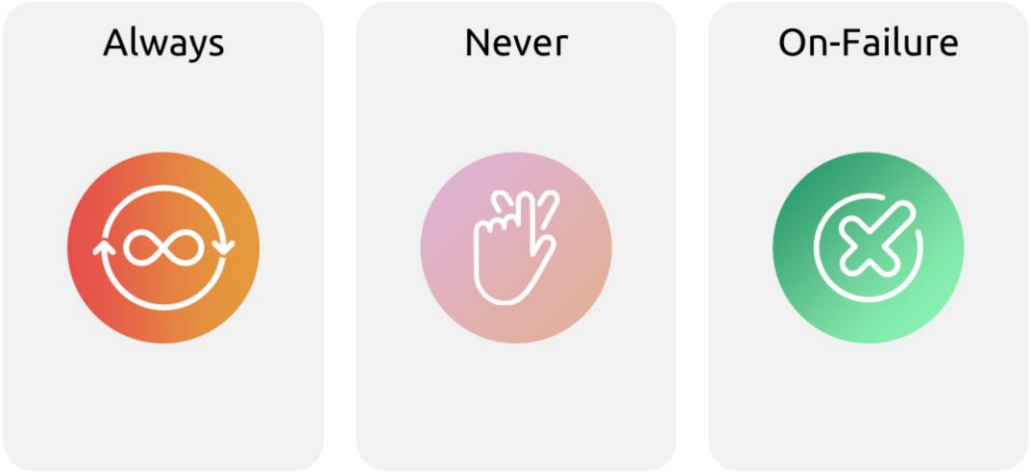
Storage
Supported volumes include:
Azure file share
Secret
Empty directory
Cloned git repo

Common scenarios
These images can then be delivered by different teams and have separate resource requirements.
Example usage could include:
A container serving a web application and a container pulling the latest content from source control.
An application container and a logging container. The logging container collects the logs and metrics output by the main application and writes them to long-term storage.
An application container and a monitoring container. The monitoring container periodically makes a request to the application to ensure that it's running and responding correctly, and raises an alert if it's not.
A front-end container and a back-end container. The front end might serve a web application, with the back end running a service to retrieve data.

# Restart Policies and Environmental Variables

# Running Containerized Tasks With Restart Policies

When you create a container group in Azure Container Instances, you can specify one of the three restart policy settings






Overview
With a configurable restart policy, you can specify that your containers are stopped when their processes have completed.

Container restart policy
Always - Containers in the container group are always restarted. This is the default setting applied when no restart policy is specified at container creation.
Never - Containers in the container group are never restarted. The containers run at most once.

OnFailure - Containers in the container group are restarted only when the process executed in the container fails (when it terminates with a nonzero exit code). The containers are run at least once.

# Setting Environment Variables in Container Instances

YAML Example


```
az container create \
    --resource-group myResourceGroup \
    --name mycontainer2 \
    --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
    --restart-policy OnFailure \
    --environment-variables 'NumWords'='5' 'MinLength'='8'
```



-Set a secure environment variable by specifying the secure Value property instead of the regular value for the variable's type.
-The two variables defined in the YAML demonstrate the two variable types.

You would deploy the YAML example file with the following command:
az container create --resource-group myResourceGroup \
  --file secure-env.yaml

# Setting Environment Variables in Container Instances

```
az container create \
    --resource-group myResourceGroup \
    --name mycontainer2 \
    --image mcr.microsoft.com/azuredocs/aci-wordcount:latest \
    --restart-policy OnFailure \
    --environment-variables 'NumWords'='5' 'MinLength'='8'
```



-Provides dynamic configuration of the application or script run by the container.
-ACI supports both Windows and Linux containers to pass secrets as environment variables
-In the example two variables are passed to the container when it is created.

# Mounting Storage to ACI

# Mounting an Azure File Share in Azure Container Instances

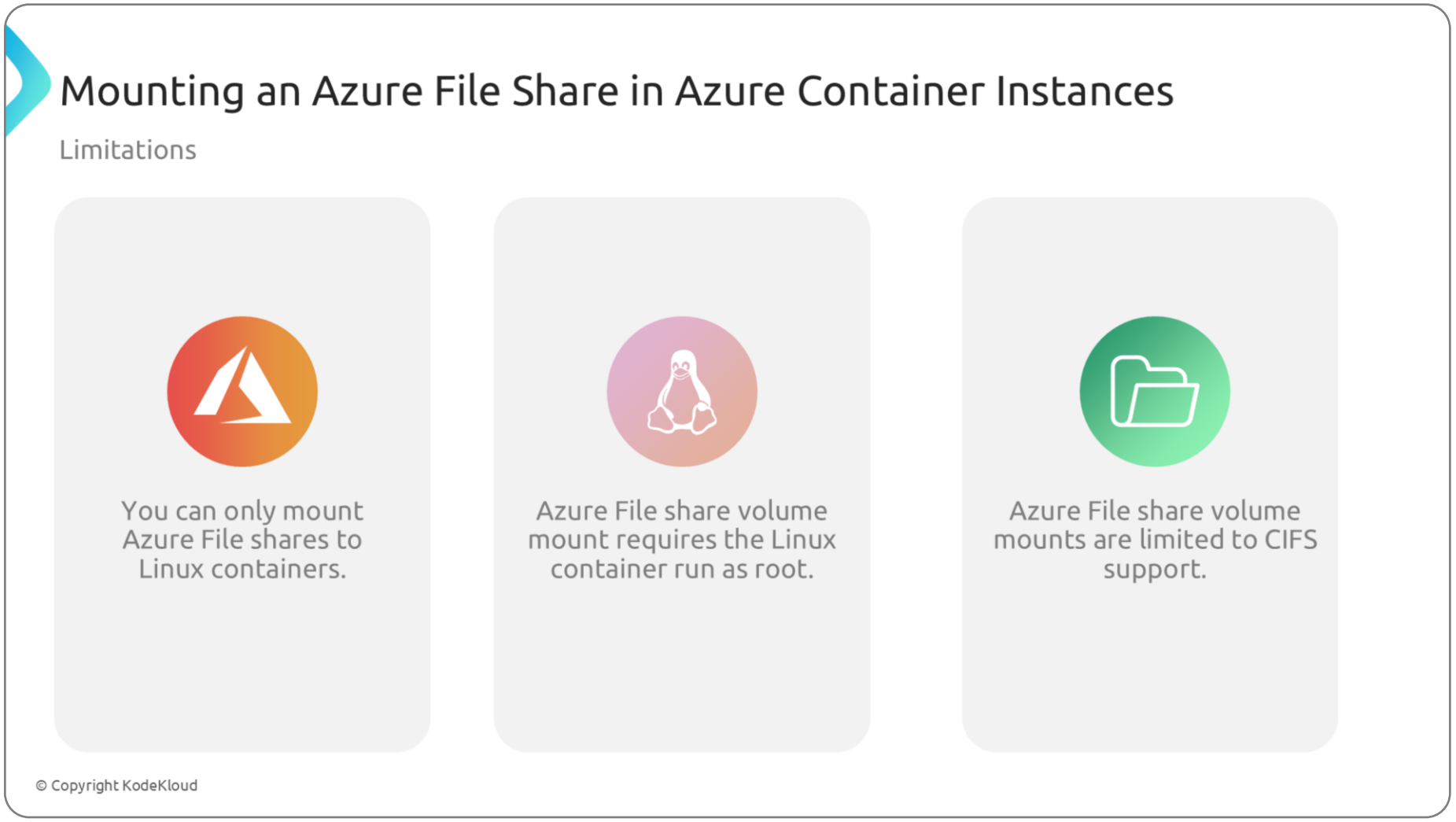
## Overview

By default, Azure Container Instances are stateless. If the container crashes or stops, its entire state is lost.

To persist the state beyond the lifetime of the container, you must mount a volume from an external store.



Overview
By default, Azure Container Instances are stateless. If the container crashes or stops, all of its state is lost.
To persist state beyond the lifetime of the container, you must mount a volume from an external store.




Deploying by YAML template is the preferred method when deploying container groups consisting of multiple containers.

Deploy container and mount volume - YAML
You can also deploy a container group and mount a volume in a container with the Azure CLI and a YAML template.

# Mounting an Azure File Share in Azure Container Instances

## Mount multiple volumes

To mount multiple volumes in a container instance, you must deploy using an Azure Resource Manager template or a YAML file.


To use a template or YAML file, provide the share details and define the volumes by populating the volumes array in the properties section of the template.



Deploying by YAML template is the preferred method when deploying container groups consisting of multiple containers.

# Implementing Azure Container Apps

# Introduction

01 Explore Azure Container Apps and their benefits

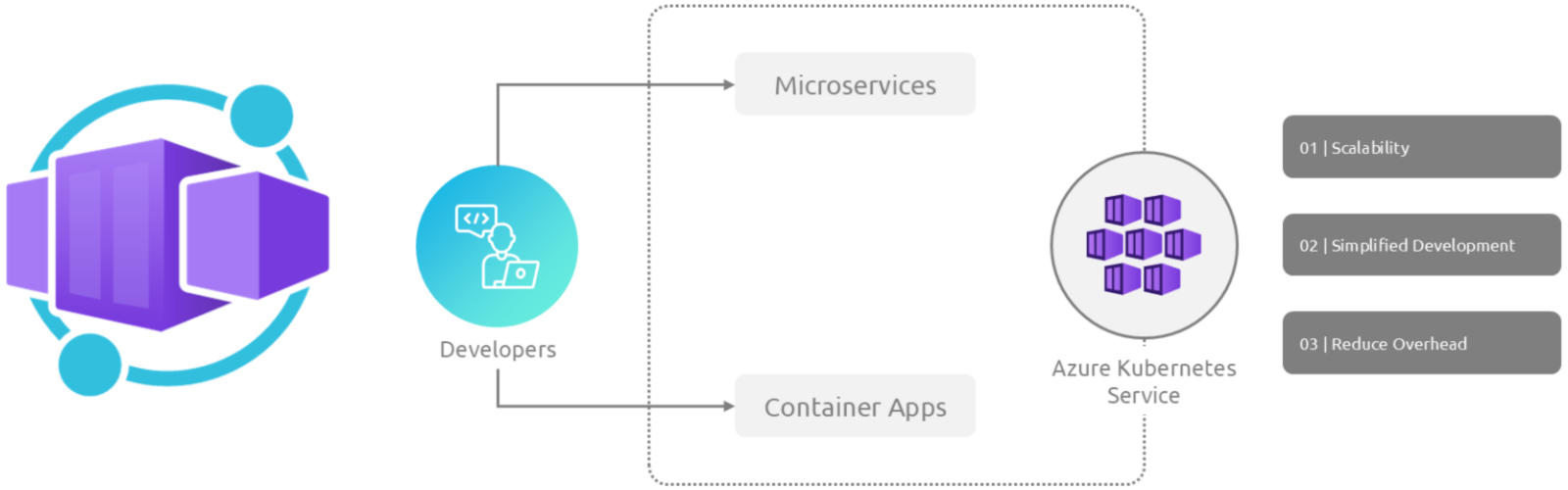
02 Deploy Azure Conatiner Apps using Azure CLI

03 Leverage built-in authentication and authorization capabilties of Azure Container Apps

04 Create revisions and implement app secrets

# Exploring Azure Container Apps

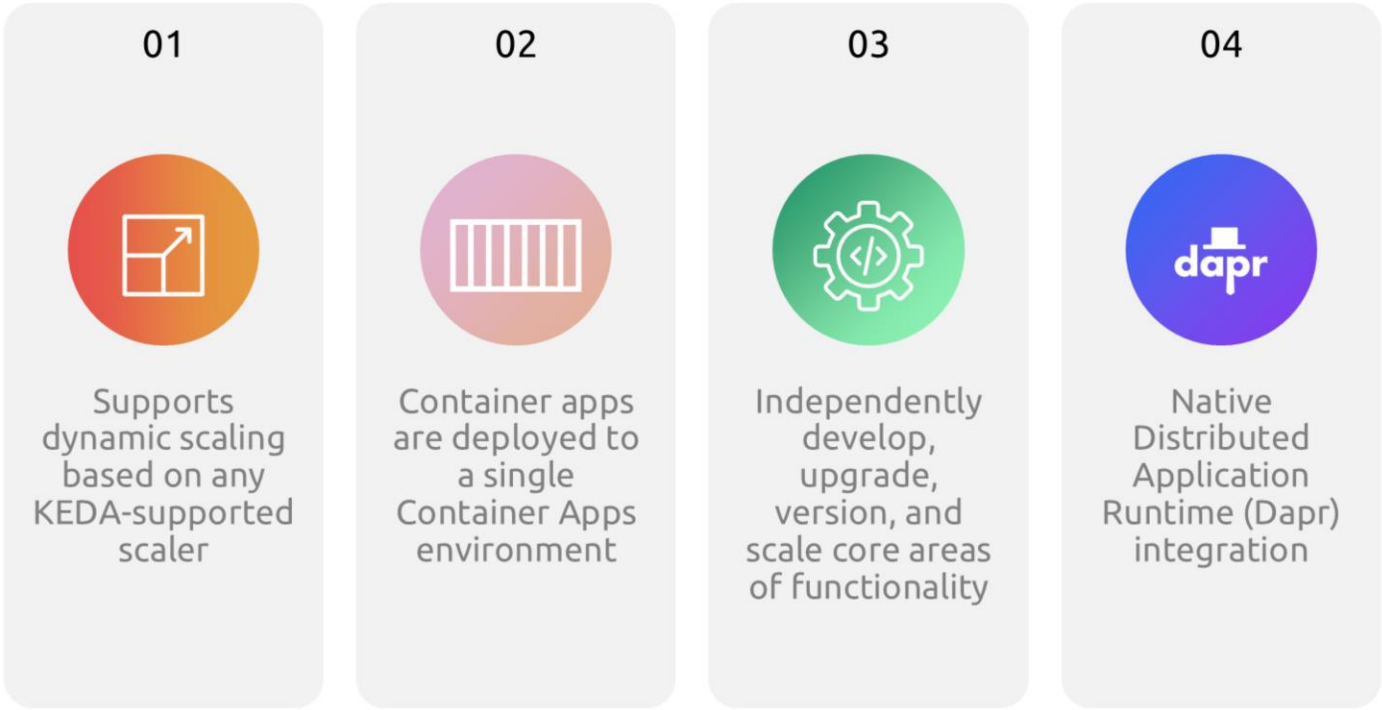
# Azure Container Apps






Exploring Azure container apps  Azure container apps is a powerful service  that allows developers to run microservices  and containerized applications or a serverless platform  This service integrates with Azure Kubernetes service  to provide scalability, simplified development  and reduced overhead  Now how this is different from Azure container instances? Container apps is actually an orchestrated,

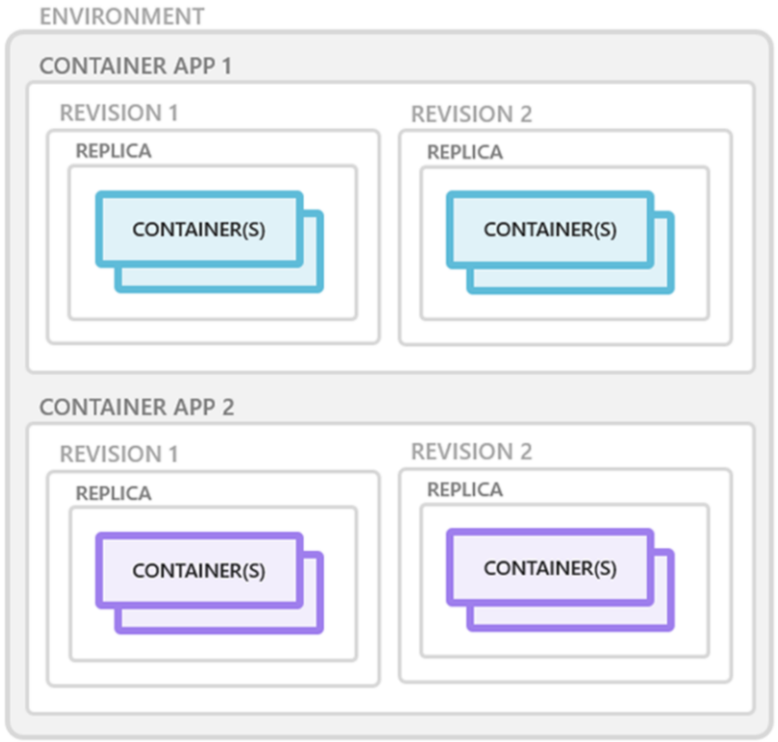
# Azure Container Apps






Azure Container Apps enables you to run microservices and containerized applications on a serverless platform that runs on top of Azure Kubernetes Service.
-Supports dynamic scaling based on any KEDA-supported scaler
-Container apps are deployed to a single Container Apps environment, which acts as a secure boundary around groups of container apps.
-Independently develop, upgrade, version, and scale core areas of functionality in an overall system.
-Native Distributed Application Runtime (Dapr) integration

# Containers in Azure Container Apps



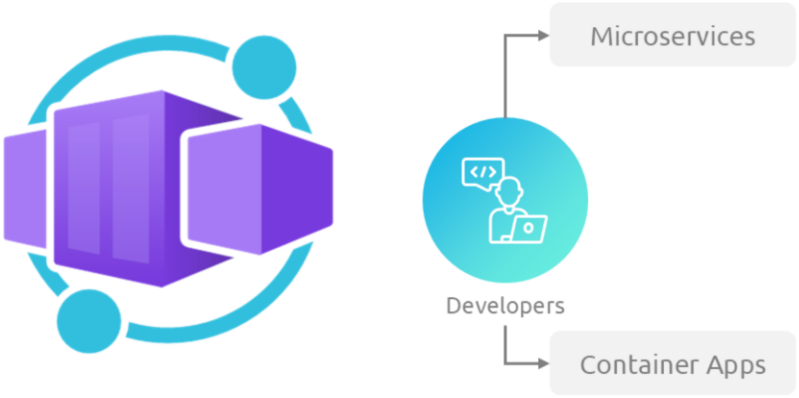



Azure Container Apps manages the details of Kubernetes and container orchestration for you. Containers in Azure Container Apps can use any runtime, programming language, or development stack of your choice.

-Containers for an Azure Container App are grouped together in pods inside revision snapshots.
-Can define multiple containers in a single container app to implement the sidecar pattern.
-Deploy images hosted on private registries by providing credentials in the Container Apps configuration.

# Azure Container Apps – Authentication and Authorization

# Azure Container Apps – Authentication and Authorization



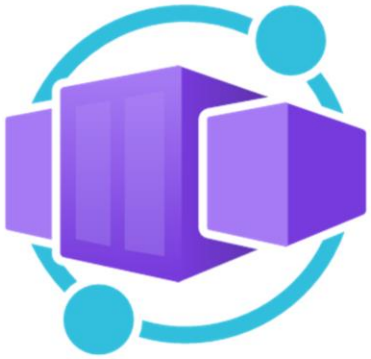



Azure Container Apps, authentication and authorization  Azure Container Apps is a powerful service  that allows developers to run microservices  and containerized applications  As we discussed earlier

# Azure Container Apps – Authentication and Authorization

Authentication

Verify identity



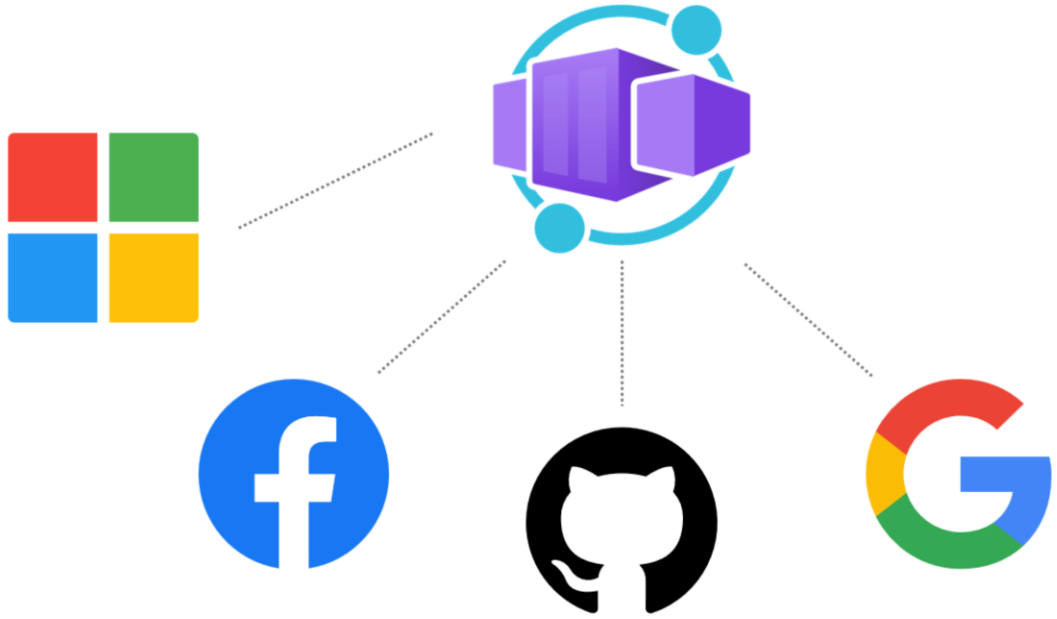
Authorization

Resource control



In this slide, we will discuss how authentication and authorization are implemented in Azure container apps Authentication is crucial for verifying the identity of users while authorization controls what resources those authenticated resource can access Container apps users federated identity

# Azure Container Apps – Authentication and Authorization

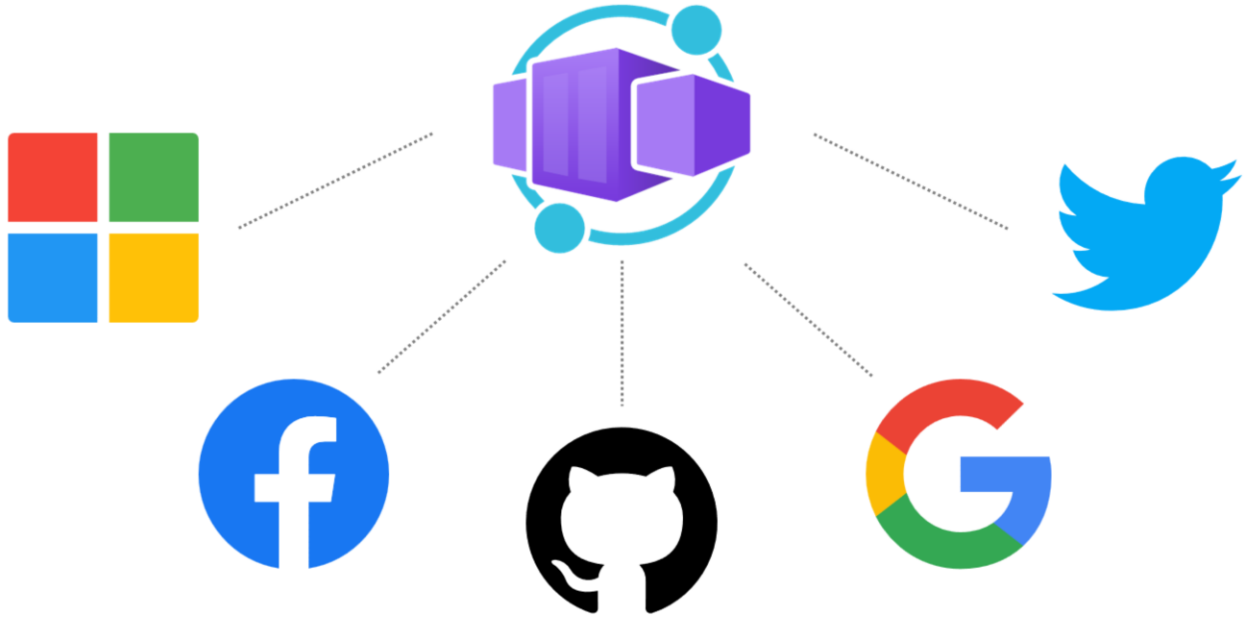


Third – Party Identity Providers



This means it allows third party identity providers, such as Microsoft Identity Platform, Facebook, GitHub, Google, and more to handle user authentication This approach removes the burden of identity management from the application developer and provides a seamless integration with existing authentication services

# Azure Container Apps – Authentication and Authorization



Popular Identity Providers by Azure Identity Platform



Some of the popular identity providers supported by Azure container apps include Azure Identity Platform, which is Azure AD or Microsoft Enter id, we have Facebook, GitHub, Google, Twitter, and any provider that supports Open Id connect

# Azure Container Apps – Authentication and Authorization

Authentication Flow is same

Difference in SDK Usage



The authentication flow is same across providers,  but differs depending on the SDK usage

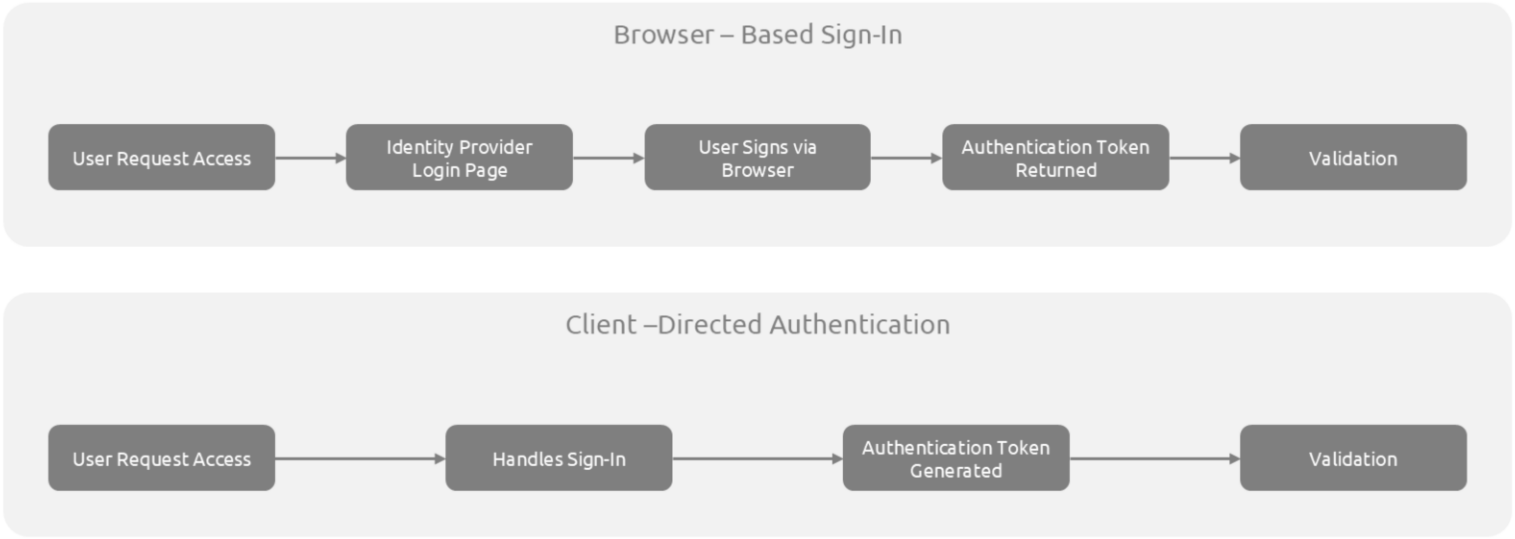
# Azure Container Apps – Authentication and Authorization

**Authentication Flow** – Without/With provider SDK



Now you can integrate this with SDK or without SDK  So when not using the provider's SDK,  the application relies on the server directed  or server flow authentication

# Azure Container Apps – Authentication and Authorization






Here the application delegates the authentication to the identity provider Typically, this involves browser based sign in where users are redirected to identity providers login page This is the same experience that we had in app service when we implemented the authentication Now if you're using the SD came, the application performs a client directed or client for authentication In this case, the application directly handles the signin on behalf of the user, and then submits the received authentication token to the container app for validation

# Managing Revisions and Secrets in Azure Container Apps

# Azure Container Apps – Revisions and Secrets

## Revisions

01

It implements container app versioning by creating revisions.

02

It controls which revisions are active and the external traffic routed to each active revision.

03

The AZ containerapp update command can modify, compute, scale, and deploy.

04

If the update includes revision scope changes, a new revision is generated.

# Azure Container Apps – Revisions and Secrets

## Secrets

Secrets are defined at the application level, and secured values are available to container apps.

Each application revision can reference one or more secrets.

When you create a container app, secrets are defined using the --secrets parameter.

# Dapr Integration With Azure Container Instances

# Dapr Integration With Azure Container Apps

| Dapr API | Description |
| --- | --- |
| Service-to-service invocation | Discover services and perform reliable, direct service-to-service calls with automatic mTLS authentication and encryption |
| State management | Provides state management capabilities for transactions and CRUD operations |
| Pub/Sub | Allows publisher and subscriber container apps to intercommunicate via an intermediary message broker |
| Bindings | Trigger your applications based on events |
| Actors | Dapr actors are message-driven, single-threaded units of work designed to quickly scale |
| Observability | Send tracing information to an Application Insights backend |
| Secrets | Access secrets from your application code or reference secure values in your Dapr components |



The Distributed Application Runtime (Dapr) provides capabilities for enabling application intercommunication, whether through messaging via pub/sub or reliable and secure service-to-service calls.

KodeKloud



Visit www.kodekloud.com to learn more.