KodeKloud

Visit www.kodekloud.com to learn more.

Exploring App Service Deployment Slots

Introduction

01 Describe Azure App Service's key components and value.

02 Explain how Azure App Service manages authentication and authorization.

03 Identify methods to control inbound and outbound traffic to your web app.

04 Deploy to an App Service using Azure CLI commands.

Azure App Service

Azure App Service

Built-in Scale Support

- Save costs and meet demand through scaling
- Scale out/in manually or automated based on metrics
- Scale up/down by changing the app service plan

Continuous Integration/ Deployment Support

- Azure DevOps
- GitHub
- Bitbucket
- FTP
- Local Git repository
- And others

Deployment Slots

- Target deployments to test or production environments
- Customize what settings are swapped between slots

Azure App Service

App Service Plans

- Define a set of compute resources
- Can run one or more apps in the same plan
- Can be scaled up or down at any time to meet compute or feature needs

Usage Tiers

- Shared: Shared compute resources targeting dev/test
- Basic: Dedicated compute targeting dev/test
- Standard: Runs production workloads
- Premium: Enhanced performance and scale
- Isolated: High performance, security, and isolation

How App Runs and Scales

- Shared: Apps receive CPU minutes on a shared VM instance and can't scale out
- Other tiers: Apps run on all VM instances configured in the App Service plan

If your app is in the same App Service plan with other apps, you may want to improve the app's performance by isolating the compute resources.

Deploying to Azure App Service

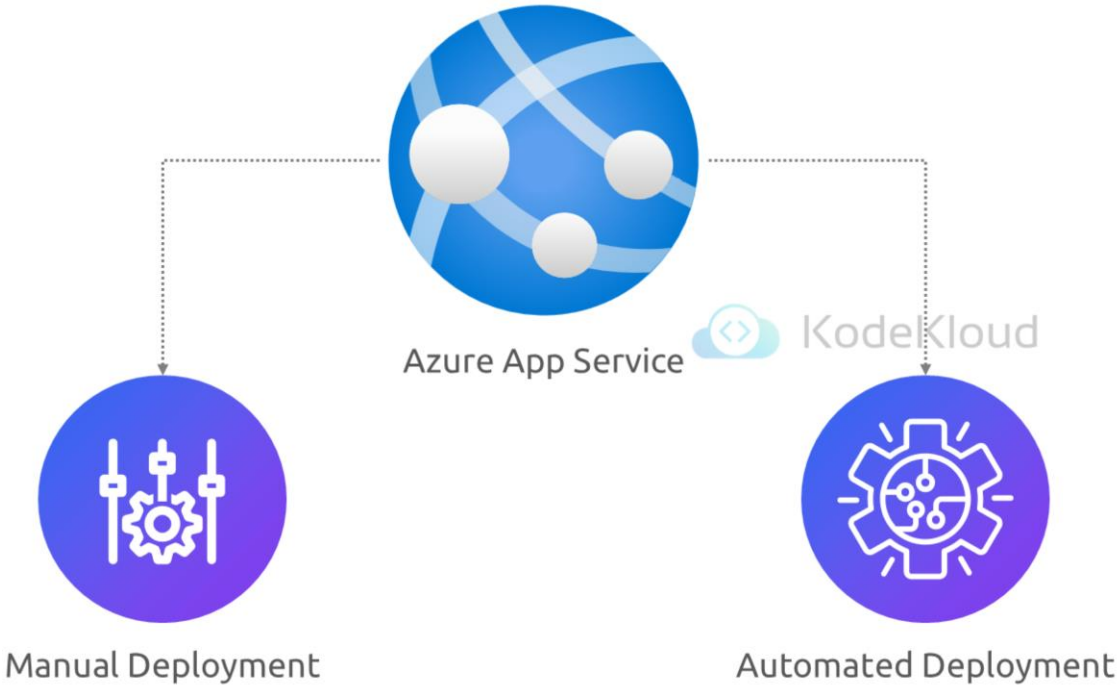
Deploying to App Service

Azure App Service

KodeKloud

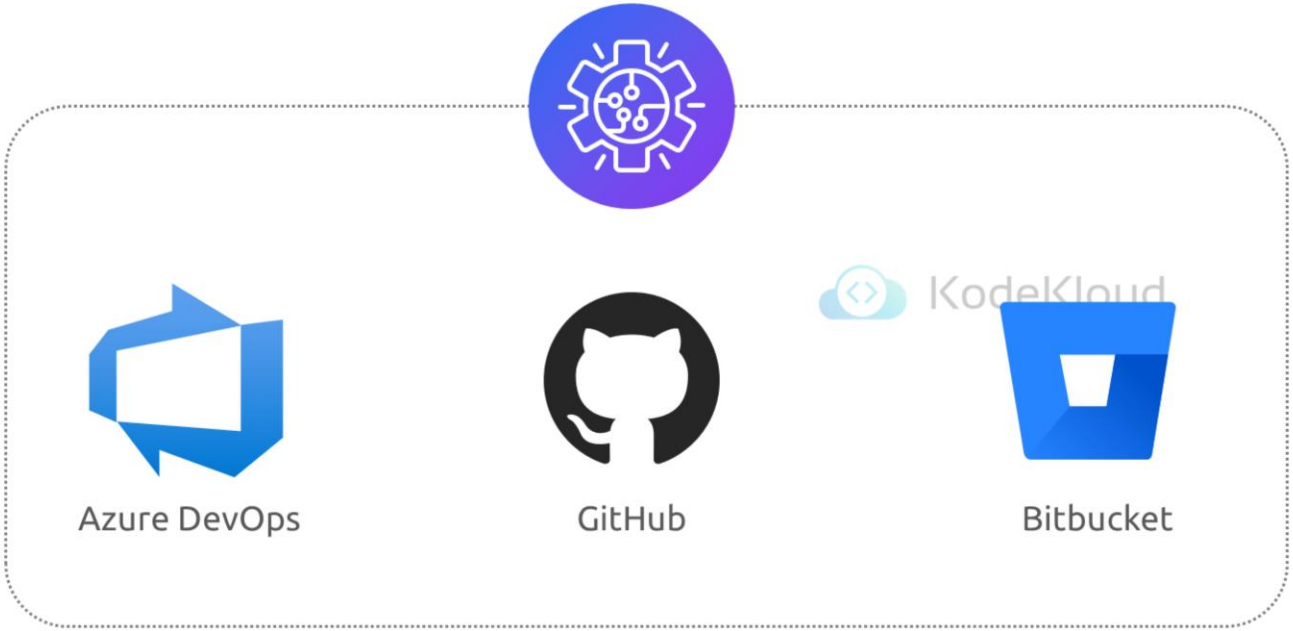
Every development team has unique requirements that can make implementing an efficient deployment pipeline difficult on any cloud service.

Deploying to App Service



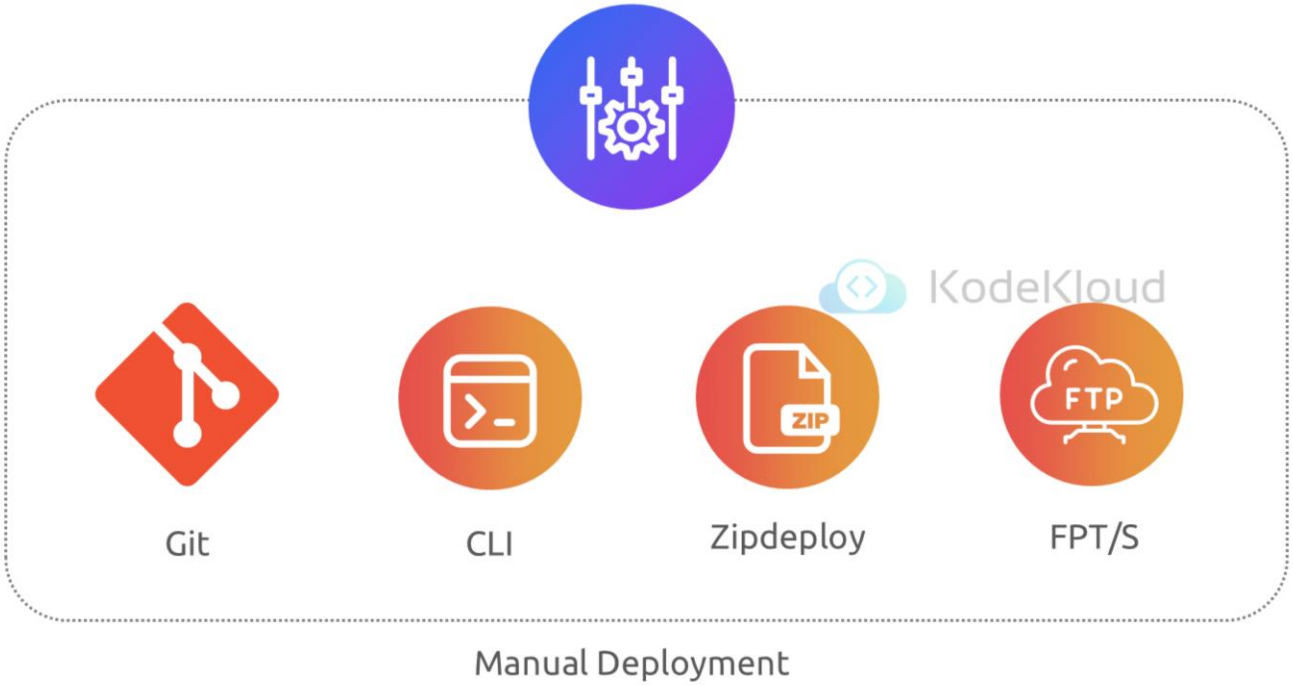
App Service supports both automated and manual deployment.

Deploying to App Service



AUTOMATED DEPLOYMENTS
Azure DevOps: You can push your code to Azure DevOps, build your code in the cloud, run the tests, generate a release from the code, and finally, push your code to an Azure Web App.
GitHub: Azure supports automated deployment directly from GitHub. When you connect your GitHub repository to Azure for automated deployment, any changes you push to your production branch on GitHub will be automatically deployed for you.
Bitbucket: With its similarities to GitHub, you can configure an automated deployment with Bitbucket.

Deploying to App Service



MANUAL DEPLOYMENTS
Git: App Service web apps feature a Git URL that you can add as a remote repository. Pushing to the remote repository will deploy your app.
CLI: webapp up is a feature of the az command-line interface that packages your app and deploys it. Unlike other deployment methods, az webapp up can create a new App Service web app for you if you haven't already created one.
Zipdeploy: Use curl or a similar HTTP utility to send a ZIP of your application files to App Service.
FTP/S: FTP or FTPS is a traditional way of pushing your code to many hosting environments, including App Service.

Azure App Service – Authentication and Authorization

App Service – Authentication and Authorization

KodeKloud

Built-in authentication and
authorization support

The authentication and authorization module run in the same sandbox as your application code. When it's enabled, every incoming HTTP request passes through it before being handled by your application code.

App Service – Authentication and Authorization

Microsoft Identity Platform

Facebook

Google

X

Apple

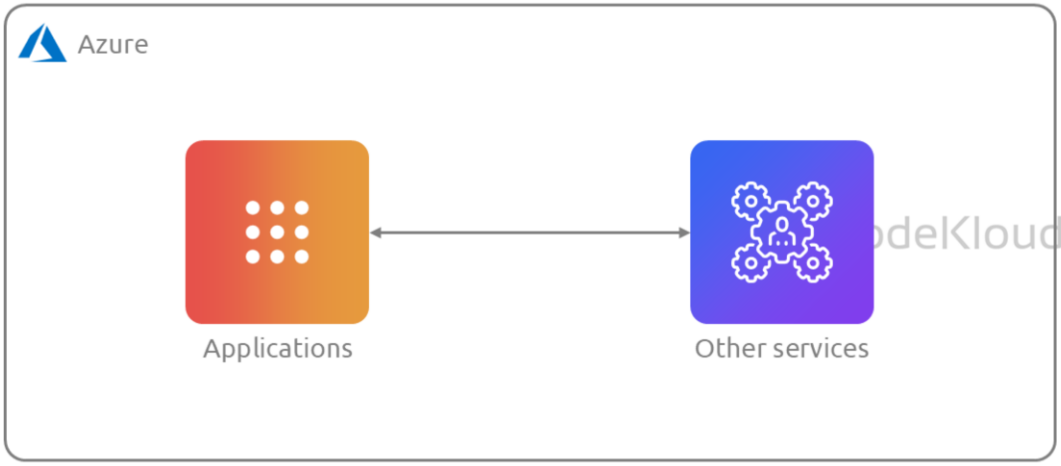
OpenID Connect

Identity providers available by default

- Microsoft Identity Platform
- Facebook
- Google
- Twitter
- Apple
- OpenID Connect

App Service Networking

App Service Networking



In this section we will explore app service networking, a critical aspect of managing how your applications interact with other services and resources within Azure.

App Service Networking

From exam standpoint, you don't have to learn the entire networking configuration, you just need to understand certain terminologies, which we will cover right now.

App Service Networking

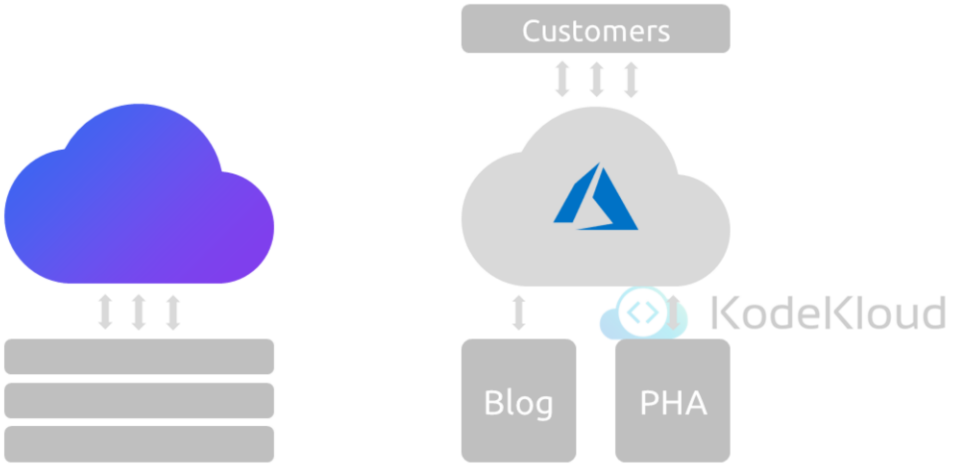
AZ 700

Apart from that, a deep dive on networking will be done in AZ 700. Another exam from Microsoft.

App Service Networking

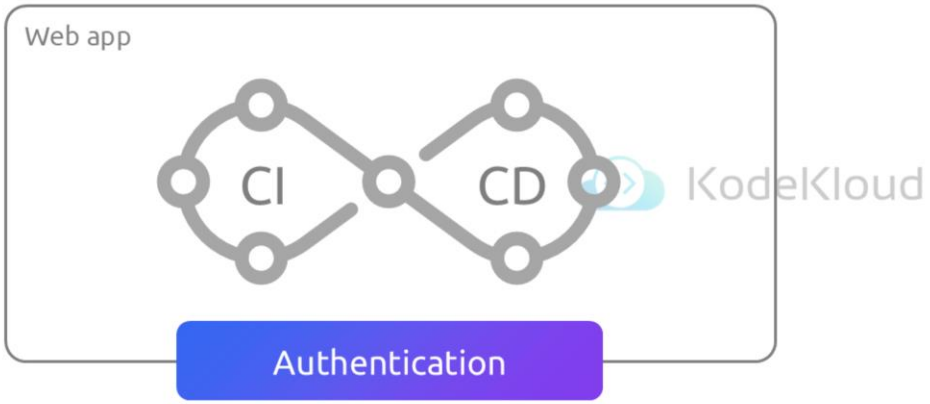
Two types of Networking:
Multi-Tenant Public Service
Single-Tenant App Service Environment

App Service Networking – Multi-tenant



So, let's start with multi-tenant public service. This model is a default for most app service plans, where multiple tenants share the same underlying infrastructure and apps are exposed to the public internet. For example, if you're running a blog site or a public PHA application, this model allows you to host your application efficiently while sharing resources with other Azure customers.

App Service Networking



In the previous demonstration, we created a web app. We tried to push the code manually using CICD, we implemented authentication. That application is an example of a multi-tenant public service because that is available to the internet, and we are also sharing the underlying resources with other customers.

App Service Networking – Single tenant

Now, when it comes to a single tenant app service environment for scenarios requiring greater isolation and security, Azure offers a dedicated environment known as App Service Environment, also known as a SC in short. So, this is a single tenant model providing dedicated resources and greater control over the networking configurations.

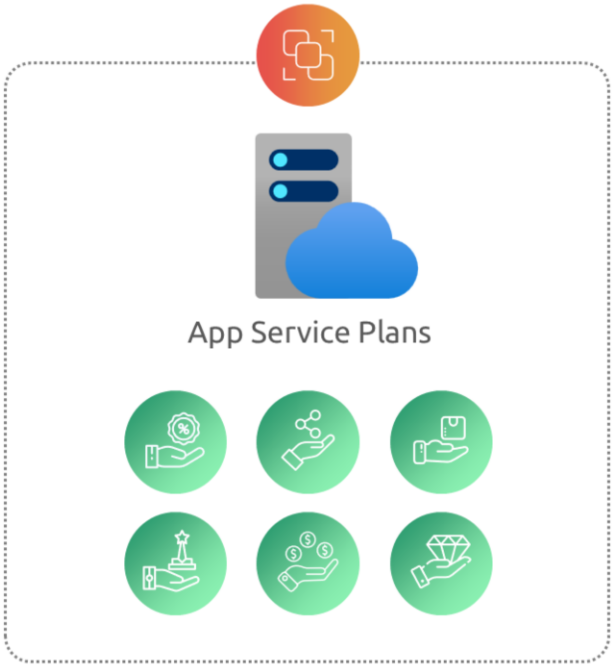
App Service Networking – Single tenant

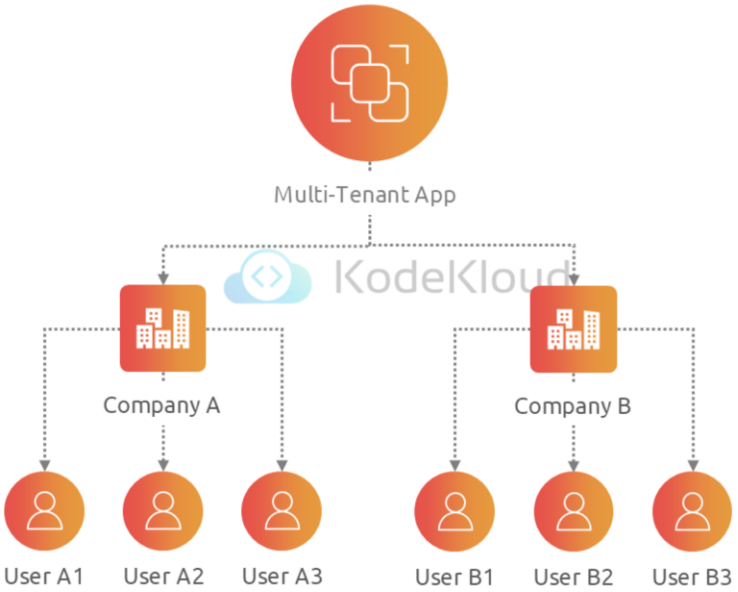
For instance, if you're operating a financial application that handles sensitive data and requires compliance with standards like GDPR, then a SC would be the ideal choice.

App Service Networking – Single tenant

So, these networking models are designed to accommodate a wide range of applications from public website to secure enterprise solutions. Let's dive deeper into these models and their specific features.

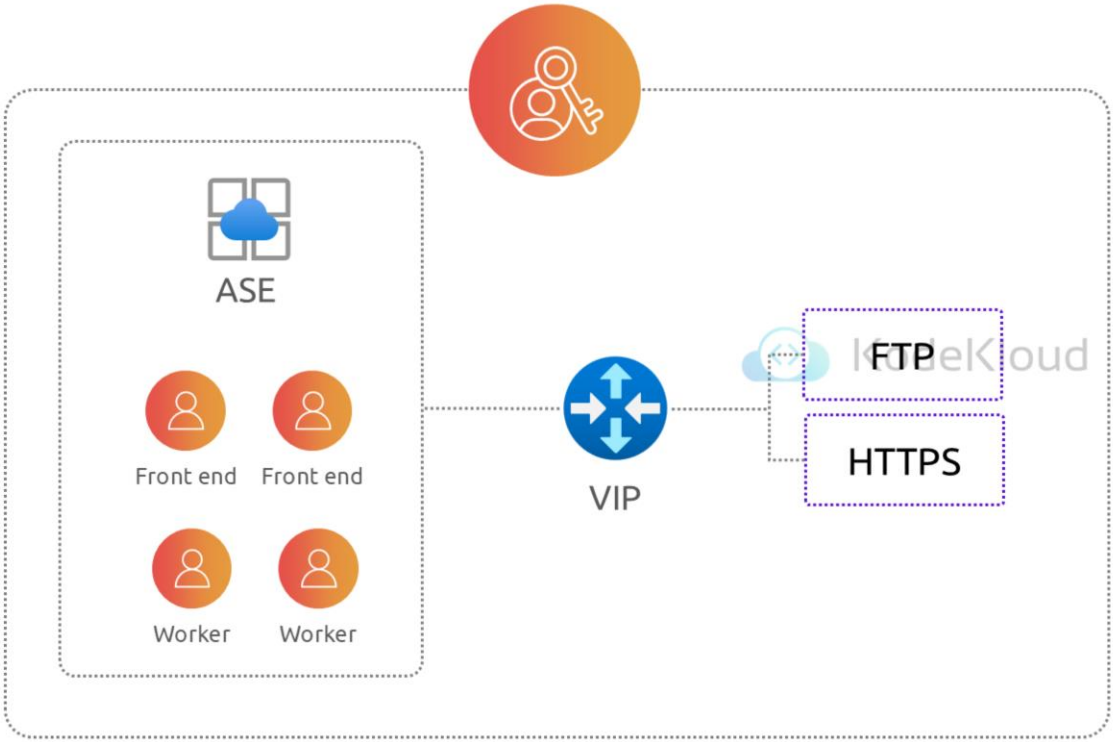
App Service Networking





Regional virtual network integration: When you connect to virtual networks in the same region, you must have a dedicated subnet in the virtual network you're integrating with.

App Service Networking



Gateway-required virtual network integration: When you connect directly to virtual networks in other regions or to a classic virtual network in the same region, you need an Azure Virtual Network gateway created in the target virtual network.

Configuring Web App Settings

Introduction

01 Configure application settings associated with deployment slots.

02 Outline methods for installing TLS certificates for your application.

03 Activate diagnostic logging to facilitate monitoring and debugging.

04 Establish virtual application to directory mappings.

Configuring Application Settings

Configure Application Settings

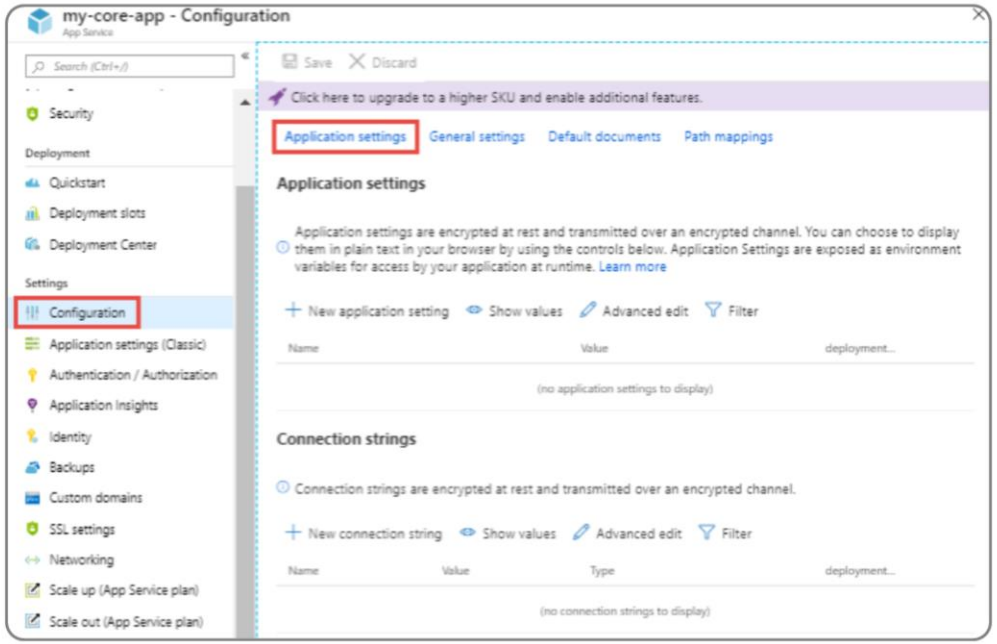
Adding and editing settings

To add a new app setting, click **New application setting.**

To add or edit app settings in bulk, click the **Advanced** edit button.

Configure connection strings

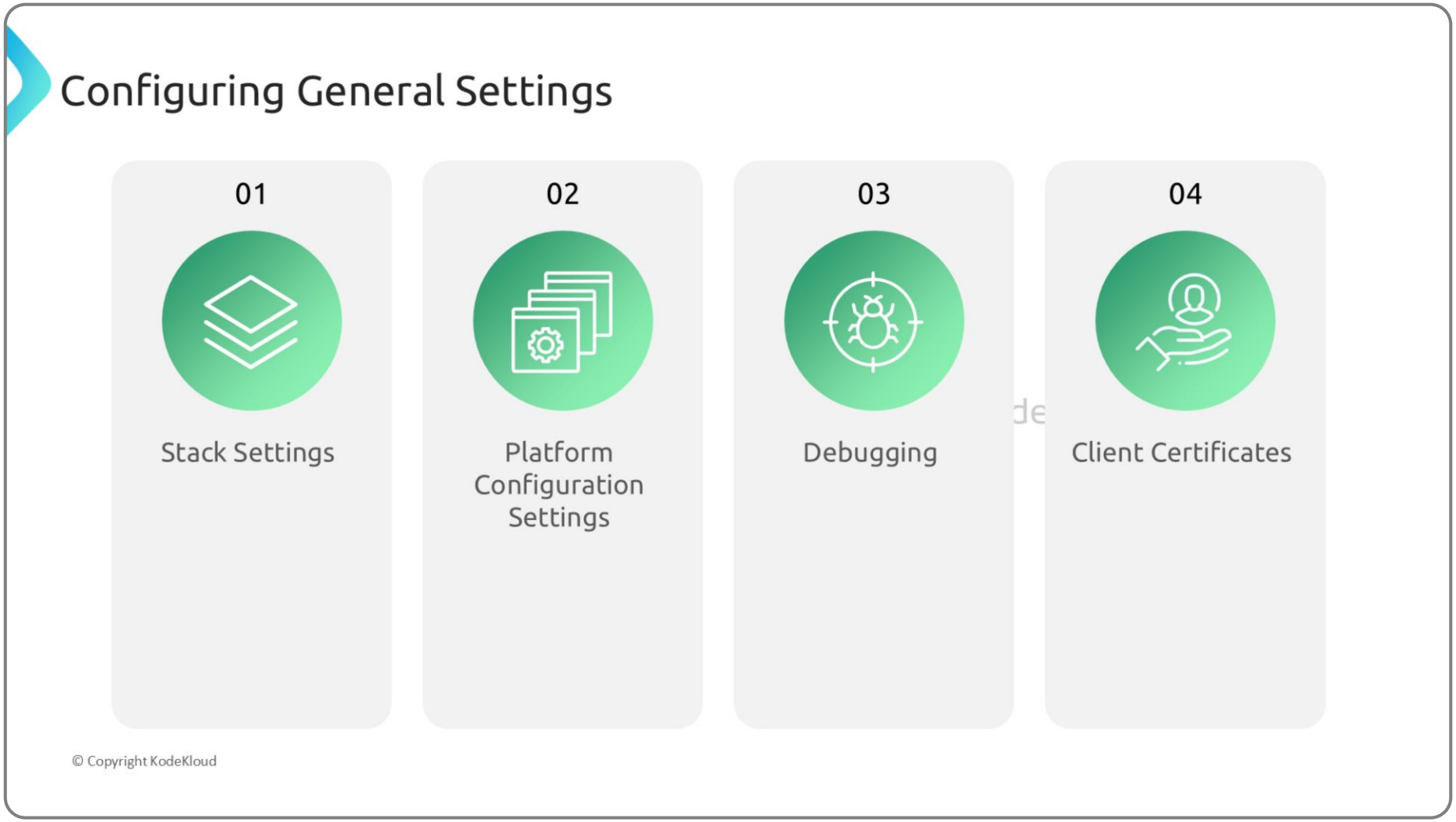
Adding and editing connection strings follow the same principles as other app settings and they can also be tied to deployment slots.



Configuring Application Settings

The highlight represents the connection string for a database. A connection string is essentially a set of parameters that an application uses to connect to a database.

Configuring General Settings



Stack settings
The software stack to run the app, including the language and SDK versions.

Platform settings
Configure settings for the hosting platform, including:

- Bitness (32-bit or 64-bit)
- WebSocket protocol

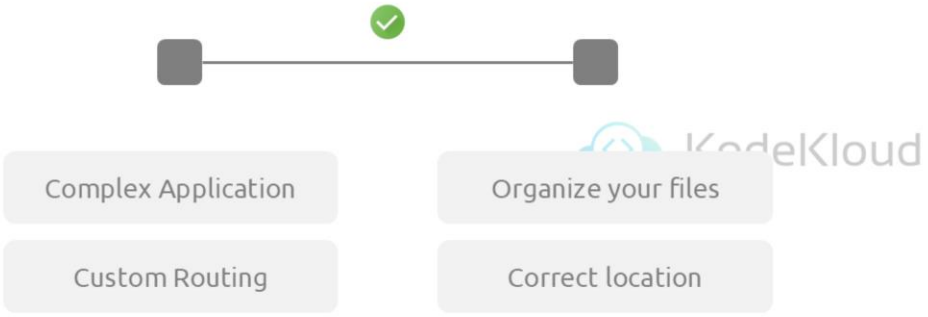
Always On
Managed pipeline version
HTTP version
ARR affinity

Debugging
Enable remote debugging for ASP.NET , ASP.NET Core, or Node.js apps.

Incoming client certificates
Require client certificates in mutual authentication. TLS mutual authentication is used to restrict access to your app by enabling different types of authentication for it.

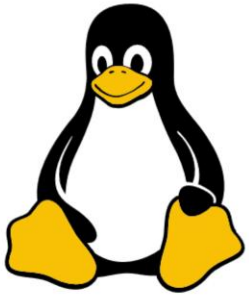
Configuring Path Mappings

Configuring Path Mappings



Path mappings are used to direct requests to the appropriate parts of your web application. This is specifically useful when you have complex applications that use multiple virtual directories or require a custom routing for specific types of content. Path mapping allows you to organize your applications files and directories more effectively, ensuring that requests are routed to the correct location.

Configuring Path Mappings



Used for Linux-based applications and containers on app services.

Example: Storing images or documents that persist beyond the container's lifecycle.

Allows adding custom or mounted storage for containerized apps.

Mount Storage refers to attaching external storage (like Azure file share) to the container for persistent data.

KodeKloud

So, to start with, let's have a look at the Linux path mapping for Linux based applications Path mappings primarily focus on contained right application This includes all Linux applications as well as Windows and Linux containers running on app service With linuxbased app, you have the ability to add a custom storage or mount storage for your containerized applications So if you go to path mappings in Azure portal, you'll find something called Mount Storage So Mount Storage refers to attaching external storage like Azure file share to your containerized application This mounted storage can be used for persistent data, such as user uploads or logs, or any data that needs to persist beyond the lifecycle of the container For instance, you could mount an Azure file share to store images or documents that your application needs to

access or store persistently

Configuring Path Mappings

Handler Mappings: Define how the server processes file types or requests (e.g., handling .php files with a PHP interpreter).

Virtual Application: A directory treated as a separate app with its own pool and settings.

Virtual Applications and Directories

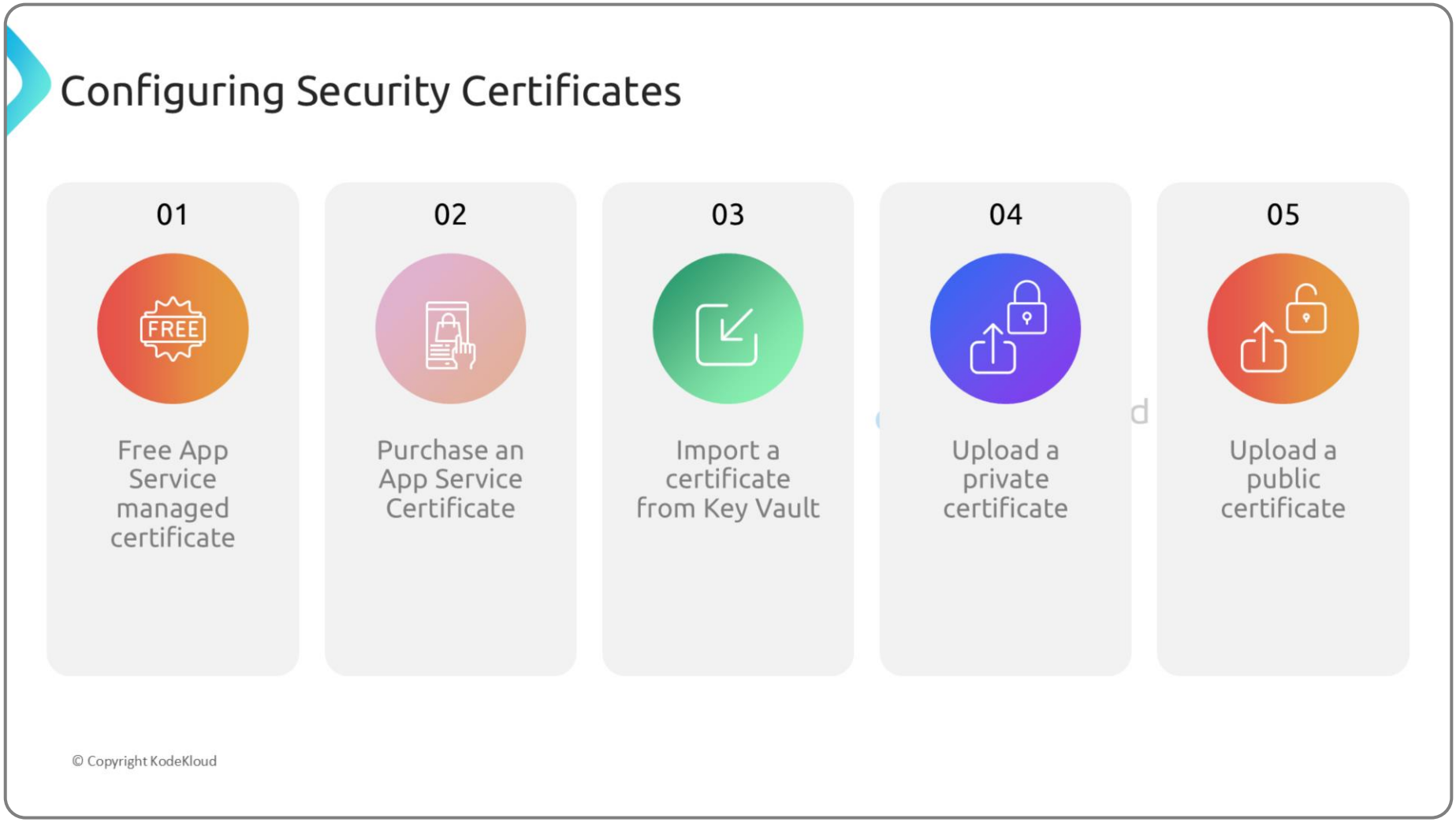
Useful for running different components (e.g., API, frontend) with separate configurations.

Virtual Directory: A non-root directory treated as part of the application.

Then we have Windows Path mappings Before explaining Windows path mapping, you need to understand two things One is the handler mappings and other one is virtual applications And directories in the context of IIS handler mapping determine how the server processes different types of files or request A handler is a piece of code that runs in response to specific type of request For example, if your application serves PHP files, IAS needs to know how to process dot php files You would configure a handler mapping to specify that request for dot PHP files should be processed by the PHP interpret This is essential for ensuring that the correct software process each file type or request Furthermore, we have virtual applications or directories A virtual directory is a directory that is not in the root of your application, but it is treated

as your application or part of your application directory For example, you might have a directory on a different drive or network share that You want to include in your website In that case, you would use a virtual directory A virtual application is a step further It is a directory that is treated as a separate application with its own application pool and settings This allows you to run different parts of your website with different configurations or even different versions of the same software This is particularly useful for large applications that have ding components like an API at a friend end that needs to be managed separately So let's go to Azure Portal and see how we can work with the path based mappings

Configuring Security Certificates



A certificate uploaded into an app is stored in a deployment unit that is bound to the app service plan's resource group and region combination (internally called a webspace). This makes the certificate accessible to other apps in the same resource group and region combination.

Notes on certificates:

Free App Service managed certificate: A private certificate that's free of charge and easy to use if you just need to secure

your custom domain in App Service.
Purchase an App Service certificate: A private certificate that's managed by Azure. It combines the simplicity of automated certificate management and the flexibility of renewal and export options.
Import a certificate from Key Vault: Useful if you use Azure Key Vault to manage your certificates.
Upload a private certificate: If you already have a private certificate from a third-party provider, you can upload it.
Upload a public certificate: Public certificates are not used to secure custom domains, but you can load them into your code if you need them to access remote resources.

Configuring Diagnostic Logging

Configuring Diagnostic Logging

Type	Platform	Description

Configuring Diagnostic Logging

Type	Platform	Description
Application logging	Windows, Linux	Log messages are generated by your application code. The messages are generated by the web framework you choose.

Configuring Diagnostic Logging

Type	Platform	Description
Application logging	Windows, Linux	Log messages are generated by your application code. The messages are generated by the web framework you choose.
Web server logging	Windows	Raw HTTP request data is in the W3C extended log file format.

Configuring Diagnostic Logging

Type	Platform	Description
Application logging	Windows, Linux	Log messages are generated by your application code. The messages are generated by the web framework you choose.
Web server logging	Windows	Raw HTTP request data is in the W3C extended log file format.
Detailed error logging	Windows	These are copies of the *.html* error pages that would have been sent to the client browser.

Configuring Diagnostic Logging

Type	Platform	Description
Application logging	Windows, Linux	Log messages are generated by your application code. The messages are generated by the web framework you choose.
Web server logging	Windows	Raw HTTP request data is in the W3C extended log file format.
Detailed error logging	Windows	These are copies of the *.html* error pages that would have been sent to the client browser.
Failed request tracing	Windows	This provides detailed tracing information on failed requests.

Configuring Diagnostic Logging

Type	Platform	Description
Application logging	Windows, Linux	Log messages are generated by your application code. The messages are generated by the web framework you choose.
Web server logging	Windows	Raw HTTP request data is in the W3C extended log file format.
Detailed error logging	Windows	These are copies of the *.html* error pages that would have been sent to the client browser.
Failed request tracing	Windows	This provides detailed tracing information on failed requests.
Deployment logging	Windows, Linux	Deployment logging happens automatically; there are no configurable settings for deployment logging.

Scaling Apps in Azure App Service

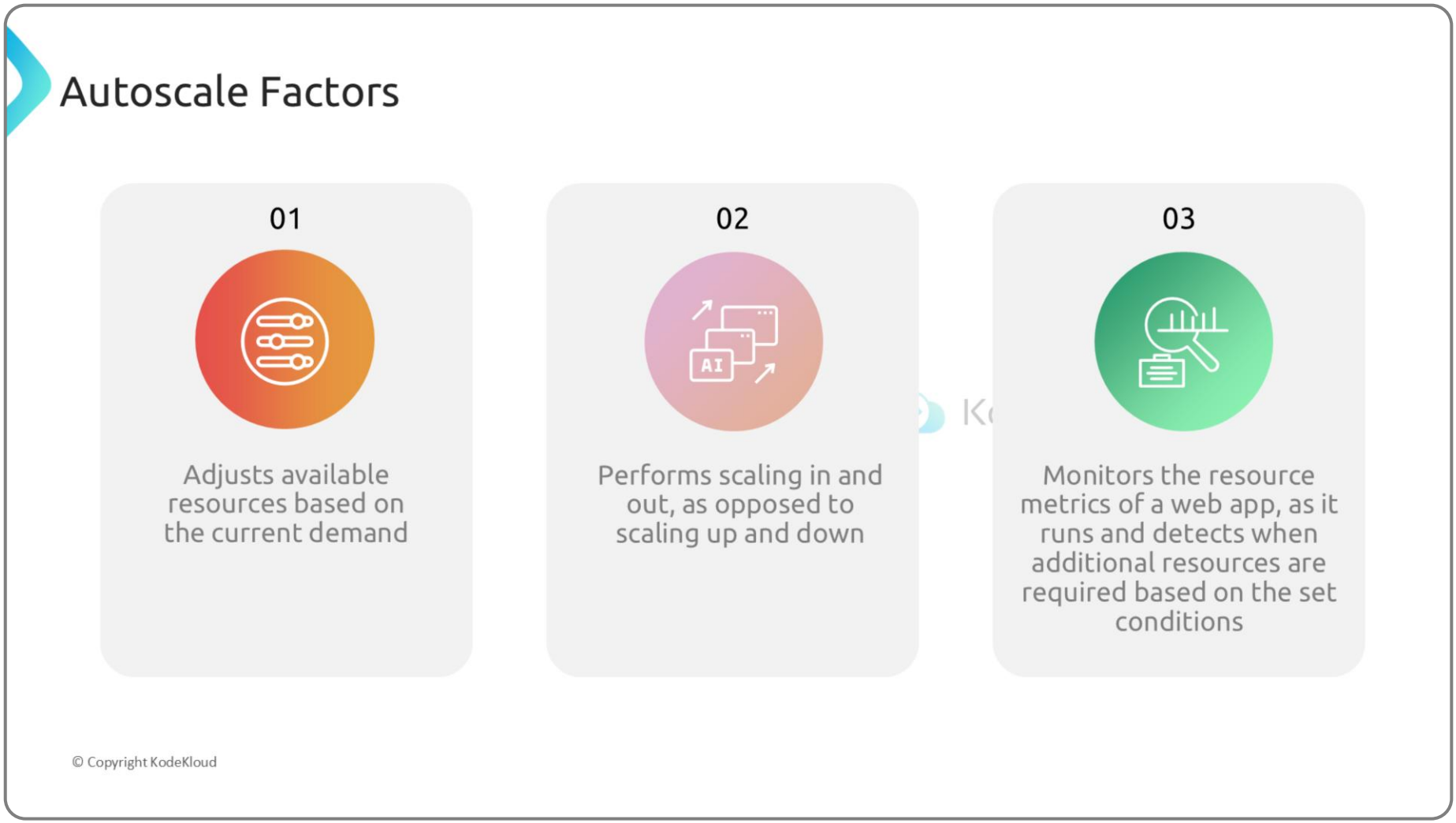
Introduction

01 Autoscaling dynamically adjusts the number of instances based on your application's demand.

02 Establish rules to define the conditions for adding or removing instances.

03 Manage and control costs effectively through precise scaling.

Examining Autoscale Factors

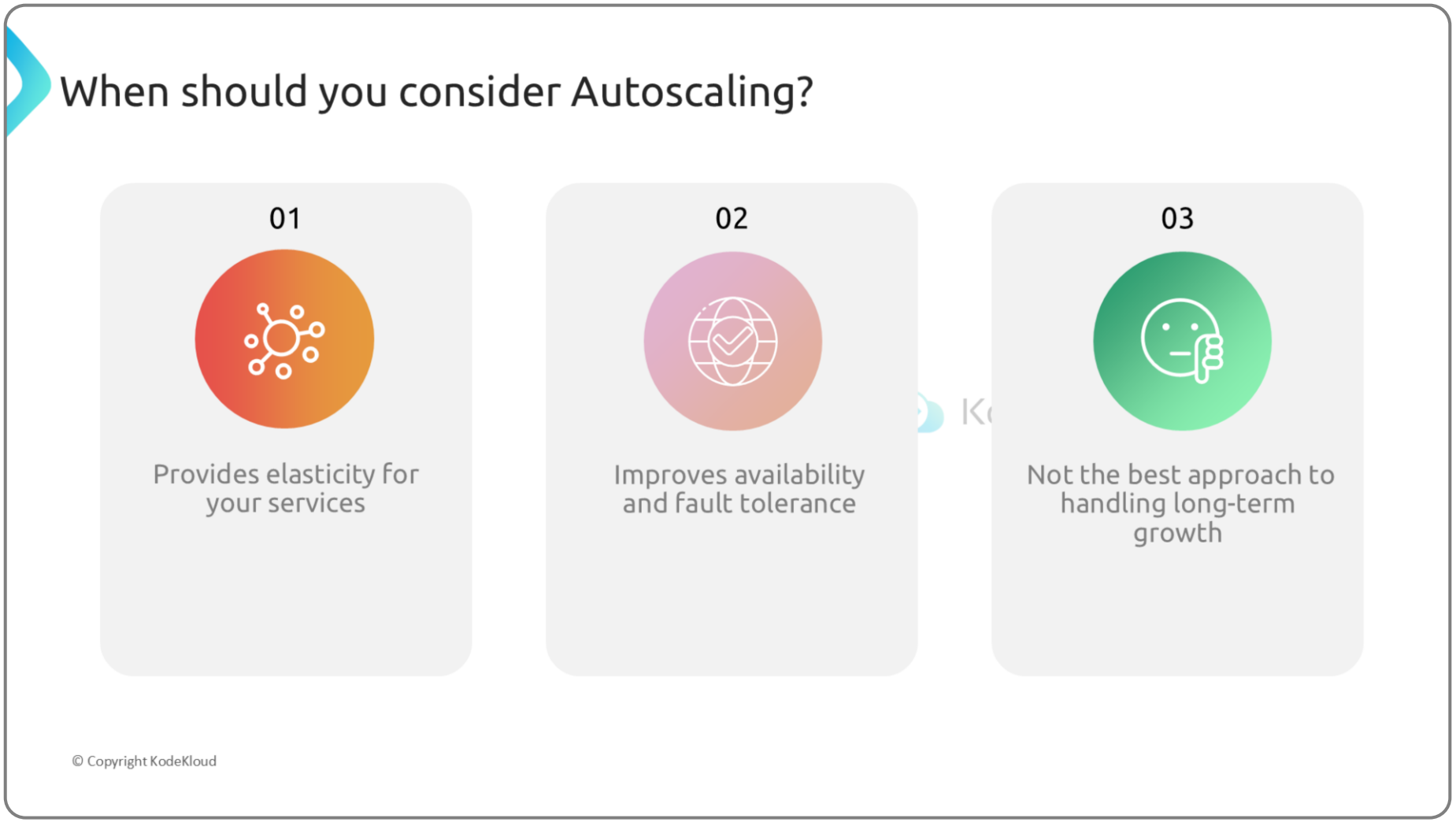


Autoscaling rules
Autoscaling makes its decisions based on rules that you define. A rule specifies the threshold for a metric, and triggers an autoscale event when this threshold is crossed. Autoscaling can also deallocate resources when the workload has diminished.

Per-app scaling can be enabled at the App Service plan level to allow for scaling an app independently from the App Service plan that hosts it. This way, an App Service plan can be scaled to 10 instances, but an app can be set to use only five. Per-app

scaling is available only for Standard, Premium, Premium V2, Premium V3, and Isolated pricing tiers.

Autoscaling has an overhead associated with monitoring resources and determining whether to trigger a scaling event. In this scenario, if you can anticipate the rate of growth, manually scaling the system over time may be a more cost effective approach.



Autoscaling rules
Autoscaling makes its decisions based on rules that you define. A rule specifies the threshold for a metric, and triggers an autoscale event when this threshold is crossed. Autoscaling can also deallocate resources when the workload has diminished.

Per-app scaling can be enabled at the App Service plan level to allow for scaling an app independently from the App Service plan that hosts it. This way, an App Service plan can be scaled to 10 instances, but an app can be set to use only five. Per-app

scaling is available only for Standard, Premium, Premium V2, Premium V3, and Isolated pricing tiers.

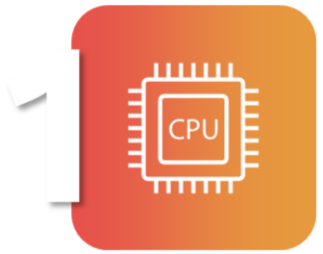
Autoscaling has an overhead associated with monitoring resources and determining whether to trigger a scaling event. In this scenario, if you can anticipate the rate of growth, manually scaling the system over time may be a more cost effective approach.

Identifying Autoscale Factors

Identifying Autoscale Factors

In this section, we will delve into the key factors that drive auto scaling decisions in Azure app service Understanding these factors will help you set up an auto scaling strategy that effectively balances performance and cost

Identifying Autoscale Factors



Scale based on a metric

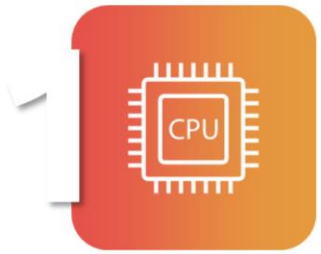
CPU Usage

Memory Consumption

Length

The first factor to consider is scaling based on a metric Common metrics include CPU usage, memory consumption, or length of a dispu

Identifying Autoscale Factors



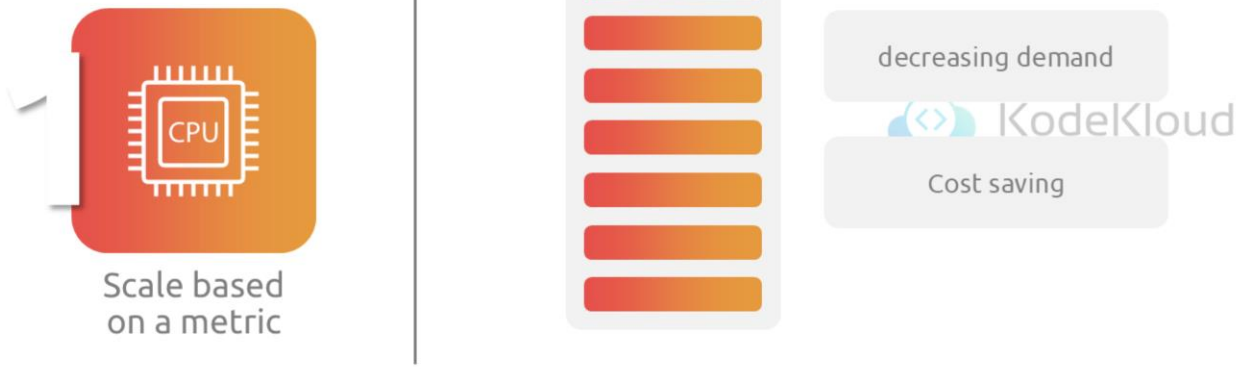
Scale based on a metric

Increasing demand

Real time metrics

For instance, you might set a rule to scale out your application by adding more instances when the CPU usage consistently exceeds % This approach ensures that your application can handle the increased demand without sacrificing performance, as it can dynamically add just resources based on the real time metrics date Similarly, once you are scaled out, let's say the demand decreases and your utilization or the CPU consumption comes below % At that point, you don't need that many instances, so we can have another rule which will reduce the number of instances In this way, you're not degrading your performance and at the same time, you're saving cost

Identifying Autoscale Factors



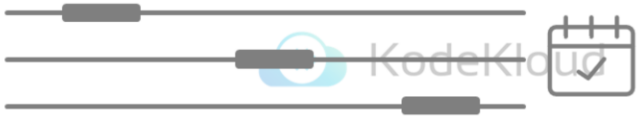
For instance, you might set a rule to scale out your application by adding more instances when the CPU usage consistently exceeds % This approach ensures that your application can handle the increased demand without sacrificing performance, as it can dynamically add just resources based on the real time metrics date Similarly, once you are scaled out, let's say the demand decreases and your utilization or the CPU consumption comes below % At that point, you don't need that many instances, so we can have another rule which will reduce the number of instances In this way, you're not degrading your performance and at the same time, you're saving cost

Identifying Autoscale Factors

Another powerful factor is scaling based on a schedule This is useful when you know in advance when demand will peak, such as during specific hours of the day or specific days of the week For example, if you run a ecommerce website that sees higher traffic on weekends, you can scale out based on a schedule, for example, on Friday evening and scale back on Sunday night This method ensures that your application is ready to handle expected traffic spikes without waiting for the metrics to trigger the scaling

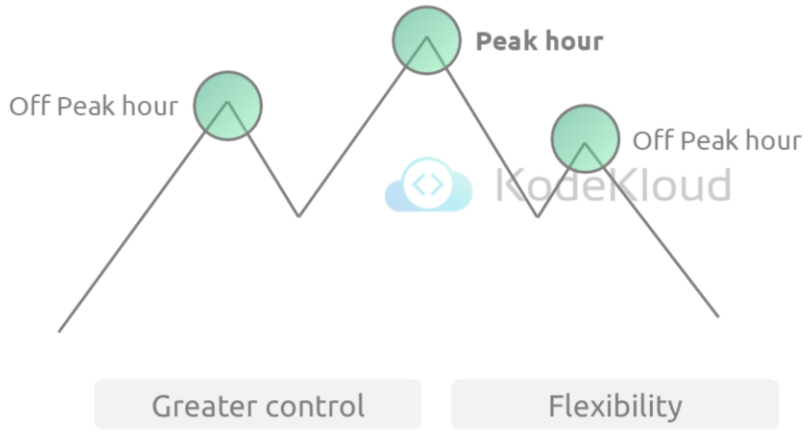
Identifying Autoscale Factors

Can create multiple conditions



Lastly, you can create multiple conditions to handle more complex auto scaling scenarios by campaigning metrics based and schedulebased tool Then you can fine tune your auto scaling strategy

Identifying Autoscale Factors



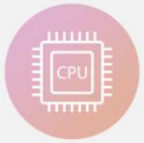
For example, you might need to scale based on CCP usage during off peak hours, but switch to a schedule based approach during peak periods This layered approach gives you greater control and flexibility, ensuring that your application remains responsive under varying conditions In the next lesson when we set up the auto scaling rule for our application, you will get to know where we can configure these values in our app service

Autoscale Metrics

01

Metrics based on all instances of an app

02



CPU percentage

03

Memory percentage

04

Disk queue length

05

HTTP queue length

06

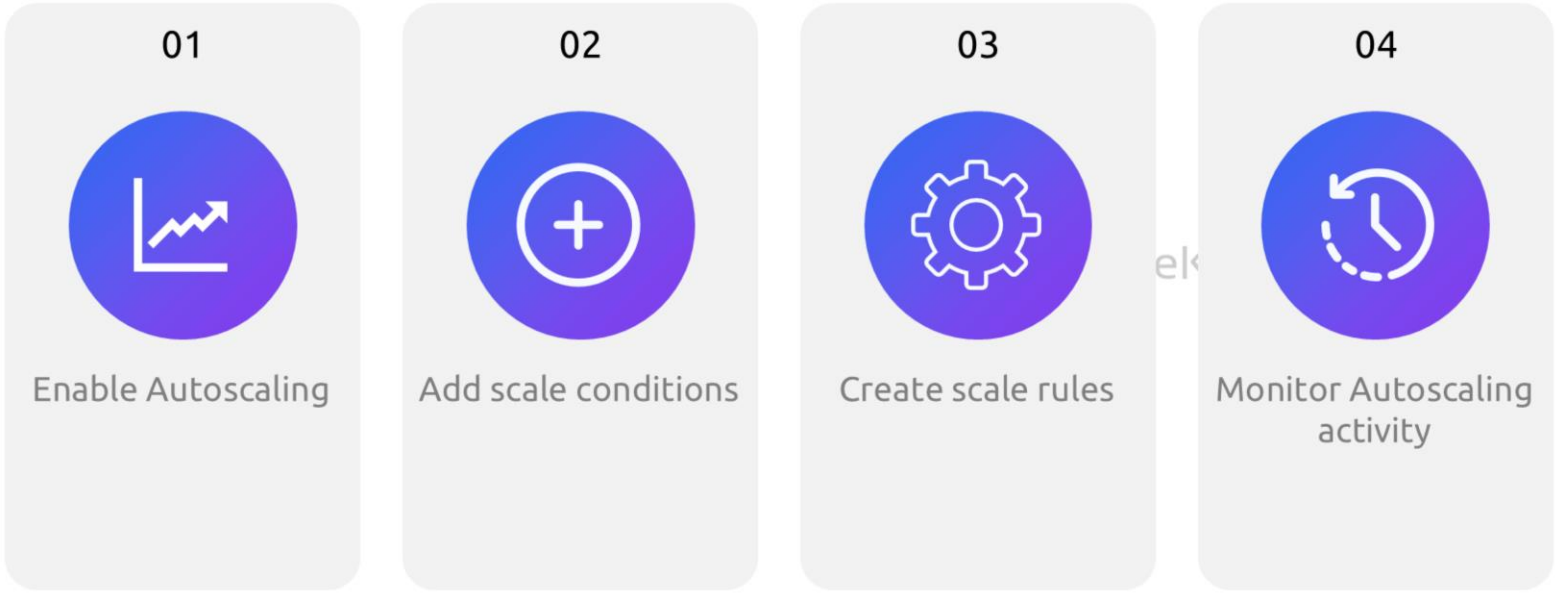
Data in - Number of bytes received

07

Data out - Number of bytes Sent

Enabling Autoscale in Azure App Service

Enabling Autoscale in Azure App Service



Enable autoscaling
- Not all pricing tiers support autoscaling
- Some are only single instance or limited to manual scaling

Add scale conditions
- A default scale condition is created
- Edit the default or create additional conditions

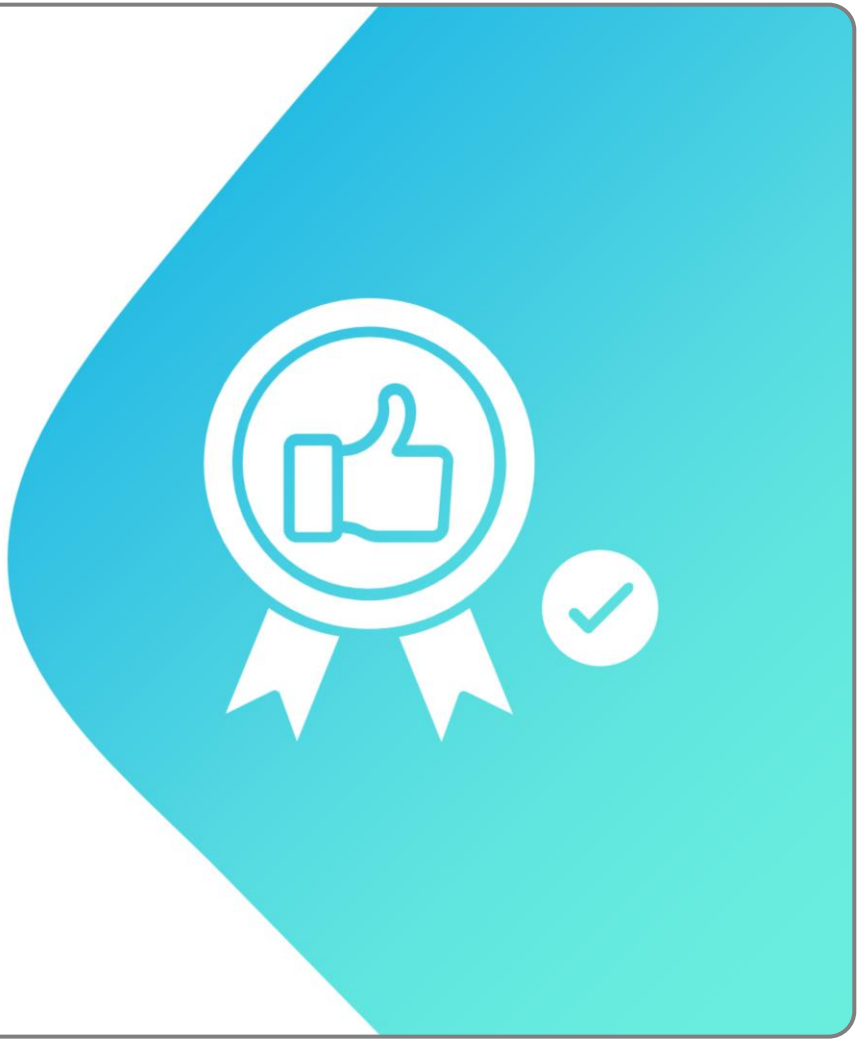
Create scale rules

- Conditions can contain one or more scale rules
- Rules can be set based on calendar, metric, or both

Monitor autoscaling behavior

- View autoscale changes on the Run history chart
- Tracks number of instances and which condition triggered the change

Autoscale – Best Practices



Autoscale – Best Practices

Ensure the maximum and minimum values are different and have an adequate margin between them

Choose the appropriate statistic for your diagnostics metric

Choose thresholds carefully for all metric types

Autoscale – Best Practices

Check for conflicts when multiple rules are configured in a condition

Always select a safe default instance count

Configure autoscale notifications

Scaling Apps in Azure App Service

Introduction

01 Deployment slots allow for the preview, management, testing, and deployment of various development environments.

02 Each deployment slot functions as a live application with its own distinct hostname.

03 Swap app content and configuration elements seamlessly between slots.

Scaling Apps in Azure App Service

Staging Environments

01

Standard

02

Premium

03

Isolated

Non-Production Slot – Benefits

01

Validates app changes in a staging deployment slot before swapping it with the production slot

02

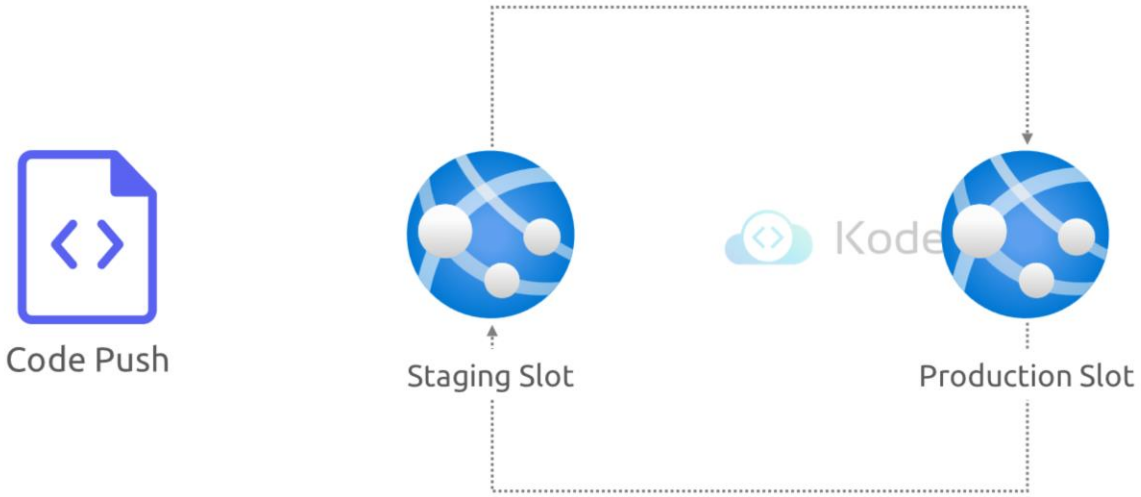
Ensures that all instances of the slot are warmed up before being swapped into production

03

After a swap, the slot with previously staged app now has the previous production app

Examining Slot Swapping

Slot Swapping



Applies the following settings from the target slot to all instances of the source slot:
Slot-specific app settings and connection strings, if applicable.
Continuous deployment settings, if enabled.
App Service authentication settings, if enabled.

Wait for every instance in the source slot to complete its restart.

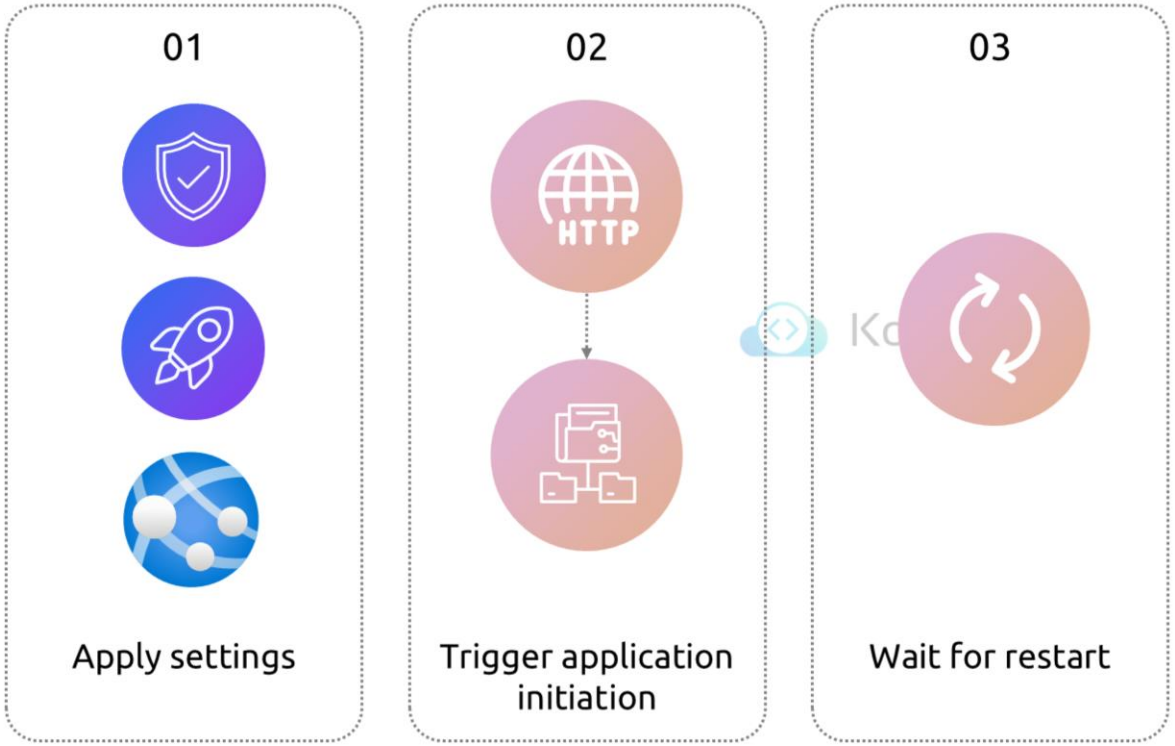
If local cache is enabled, trigger local cache initialization by making an HTTP request to the application root ("/") on each instance of the source slot.

If auto swap is enabled with custom warm-up, trigger Application Initiation by making an HTTP request to the application root ("/") on each instance of the source slot.

If all instances on the source slot are warmed up successfully, swap the two slots by switching the routing rules for the two slots.

Now that the source slot has the pre-swap app previously in the target slot, perform the same operation by applying all settings and restarting the instances.

Slot Swapping



Applies the following settings from the target slot to all instances of the source slot:
Slot-specific app settings and connection strings, if applicable.
Continuous deployment settings, if enabled.
App Service authentication settings, if enabled.

If auto swap is enabled with custom warm-up, trigger Application Initiation by making an HTTP request to the application root ("/") on each instance of the source slot.

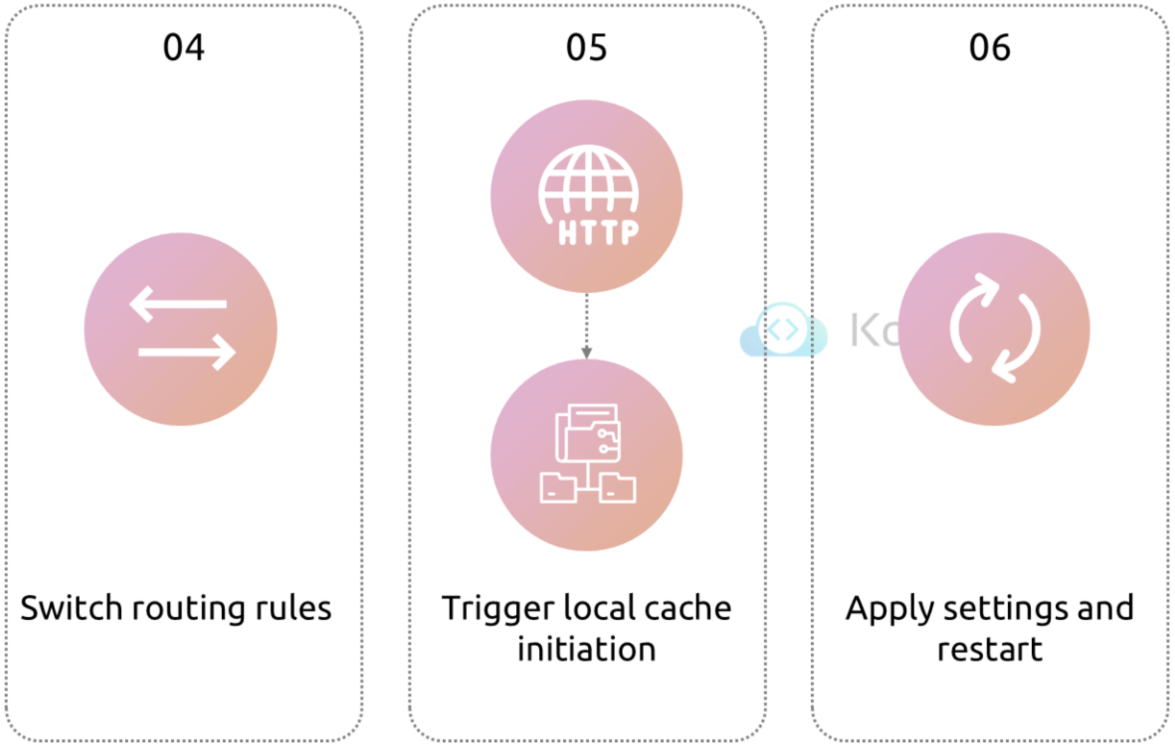
Wait for every instance in the source slot to complete its restart.

If all instances on the source slot are warmed up successfully, swap the two slots by switching the routing rules for the two slots.

If local cache is enabled, trigger local cache initialization by making an HTTP request to the application root ("/") on each instance of the source slot.

Now that the source slot has the pre-swap app previously in the target slot, perform the same operation by applying all settings and restarting the instances.

Slot Swapping



Applies the following settings from the target slot to all instances of the source slot:
Slot-specific app settings and connection strings, if applicable.
Continuous deployment settings, if enabled.
App Service authentication settings, if enabled.

If auto swap is enabled with custom warm-up, trigger Application Initiation by making an HTTP request to the application root ("/") on each instance of the source slot.

Wait for every instance in the source slot to complete its restart.

If all instances on the source slot are warmed up successfully, swap the two slots by switching the routing rules for the two slots.

If local cache is enabled, trigger local cache initialization by making an HTTP request to the application root ("/") on each instance of the source slot.

Now that the source slot has the pre-swap app previously in the target slot, perform the same operation by applying all settings and restarting the instances.

Swapping Deployment Slots

Swapping Deployment Slots

01

Swap deployment slots on your app's Deployment slots page and Overview page

02

Configure auto swap

03

Swap with preview

04

Specify a custom warm up

05

Roll back and monitor a swap

Routing Traffic in Slots

Routing Traffic in App Service

- Go to your app's resource page and select Deployment slots.
- In the Traffic % column of the slot you want to route to, specify a percentage (between 0 and 100) to represent the amount of total traffic you want to route.

V/S

- In addition to automatic traffic routing, App Service can route requests to a specific slot.
- This is useful when you want your users to be able to opt in to or opt out of your beta app.
- To route production traffic manually, use the x-ms-routing-name query parameter.

By default, all client requests to the app's production URL (http://<app_name>.azurewebsites.net) are routed to the production slot.

You can route a portion of the traffic to another slot.

This feature is useful if you need user feedback for a new update, but you're not ready to release it to production.

KodeKloud

Visit www.kodekloud.com to learn more.