Microsoft

# Module 01: Creating Azure App Service Web Apps

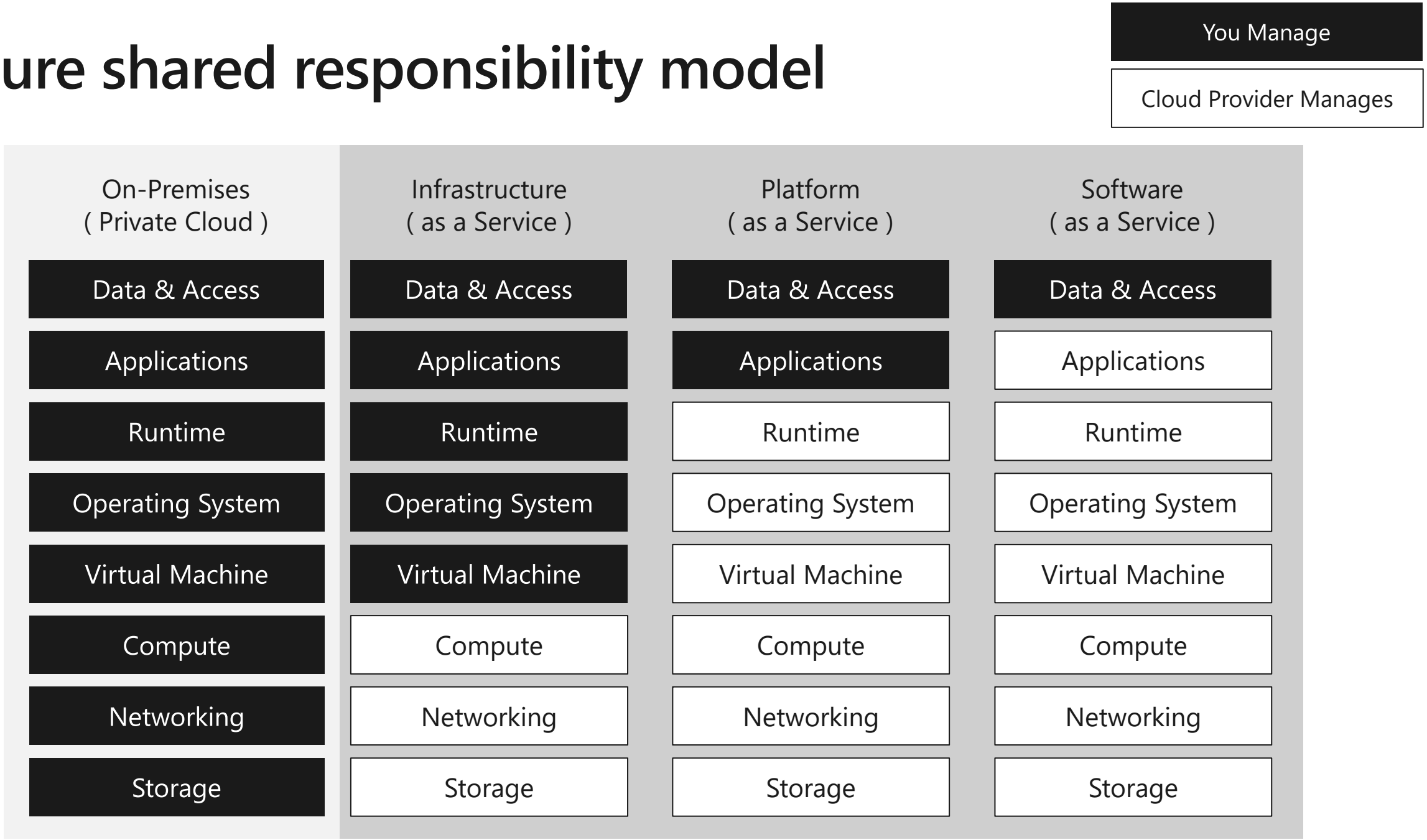
Microsoft

# Topics

- Azure App Service core concepts
- Creating an Azure App Service Web App
- Configuring and Monitoring App Service apps
- Scaling App Service apps
- Azure App Service staging environments

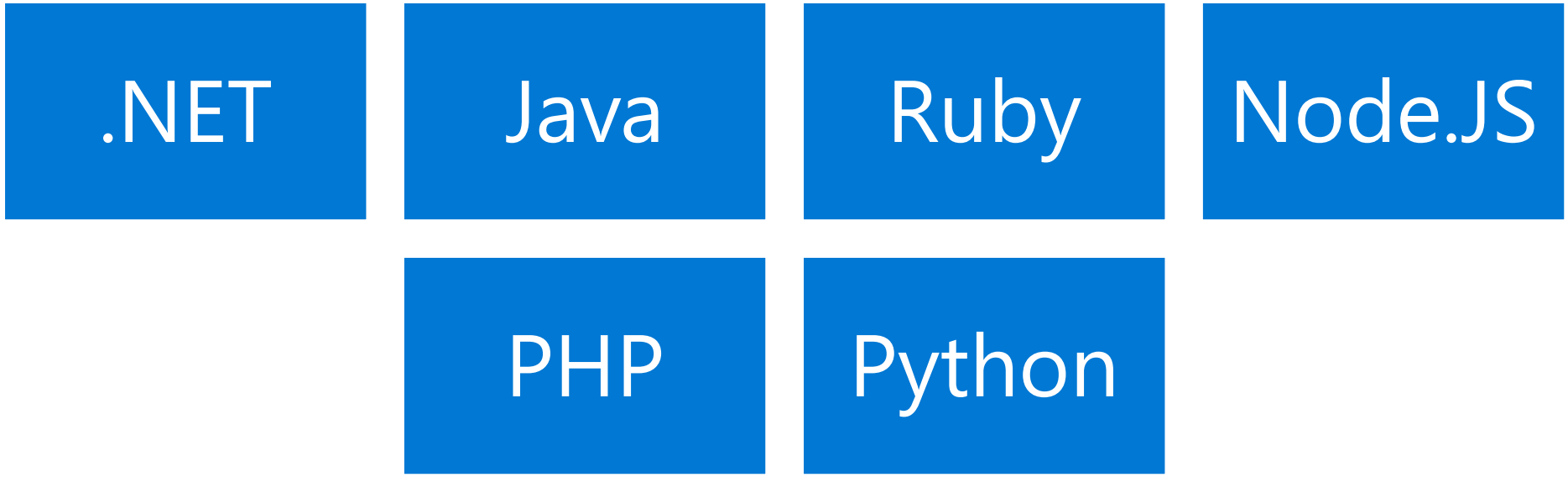
# Lesson 01: Azure App Service core concepts

# Azure shared responsibility model



# App Service

- Service for hosting web applications, REST APIs, and mobile backends can be developed in many of the following languages:




- Applications can execute and scale in a fully managed, sandbox environment

# Web Apps

- Scalable hosting for web applications:
  - Provides a quick way to host your web application in the cloud
  - Allows you to scale your web app without being required to redesign for scalability
  - Integrates with Visual Studio
  - Provides an open platform for many different programming languages
- Advantages:
  - Near instant deployment
  - SSL and Custom Domain Names available in some tiers
  - WebJobs provide background processing for independent scaling
  - Can scale to larger machines without redeploying applications

# Key features of App Service Web Apps

- Multiple languages and frameworks:
  - First-class support for Microsoft ASP.NET, Java, Ruby, Node.js, PHP, or Python
- DevOps optimization:
  - Continuous integration and deployment with Visual Studio Team Services, GitHub, Bitbucket, Docker Hub, or Azure Container Registry
- Global scale with high availability:
  - Scale up or out manually or automatically. Host anywhere in the Microsoft global datacenter infrastructure
- Connections to SaaS platforms and on-premises data:
  - More than 50 connectors for enterprise systems (such as SAP), SaaS services (such as Salesforce), and internet services (such as Facebook)

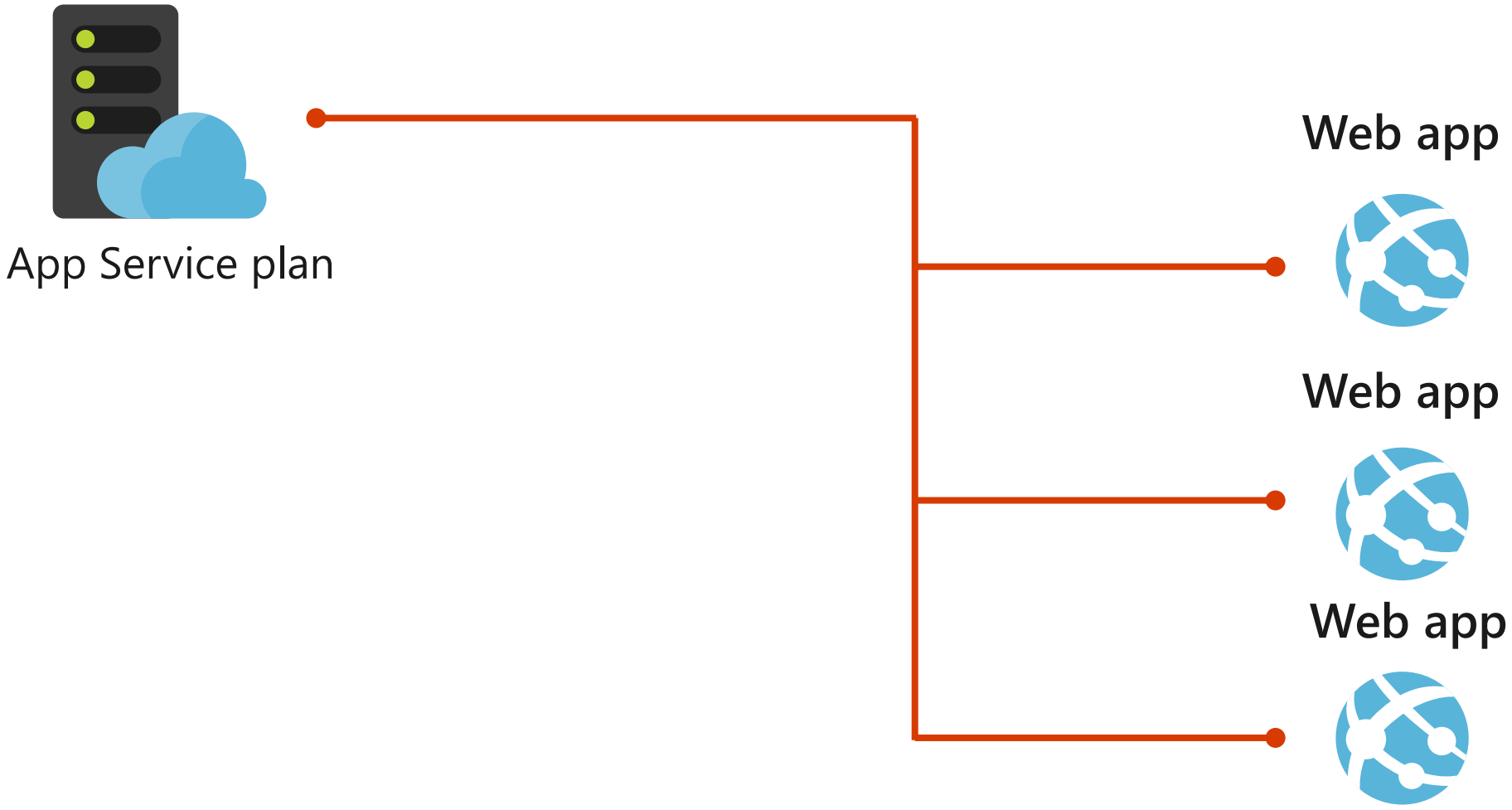
# Key features of App Service Web Apps (cont.)

- Security and compliance:
  - App Service is ISO, SOC, and PCI compliant
- Application templates:
  - Templates in the Azure Marketplace, such as WordPress, Joomla, and Drupal
- Visual Studio integration:
  - Streamline the work of creating, deploying, and debugging
- API and mobile features:
  - Turn-key Cross-Origin Resource Sharing (CORS) support for RESTful API scenarios, and enables authentication, offline data sync, push notifications, and more
- Serverless code:
  - Run code on-demand without having to explicitly provision or manage infrastructure

# App Service plans

- App Service plans can logically group apps within a subscription:
  - Characteristics such as features, capacity, and tiers are shared among the website instance in the group
  - The App Service plan is the unit of billing in most cases
- Multiple App Service plans can exist in a single Resource Group and multiple apps can exist in a single App Service plan

# App Service plans (continued)

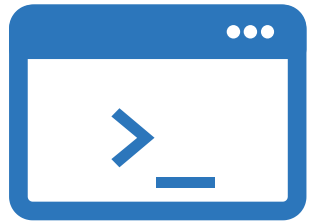


# Lesson 01: Creating an Azure App Service Web App

# Demonstration: Creating a web app by using the Azure portal

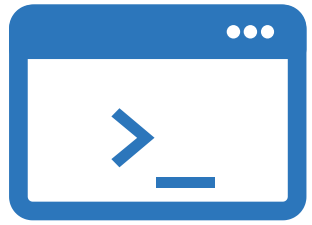
# Creating a web app with Azure command-line interface (CLI) (continued)

```
# generate a unique name and store as a shell variable
webappname=mywebapp$RANDOM

# create a resource group
az group create --location westeurope --name myResourceGroup

# create an App Service plan
az appservice plan create --name $webappname --resource-group myResourceGroup --sku
FREE

# create a Web App
az webapp create --name $webappname `
    --resource-group myResourceGroup `
    --plan $webappname
```



# Deploying a web app with Azure CLI

```
# store a repository url as a shell variable
gitrepo=https://github.com/Azure-Samples/php-docs-hello-world

# deploy code from a Git repository
az webapp deployment source config --name $webappname --resource-group myResourceGroup
--repo-url $gitrepo --branch master --manual-integration

# print out the FQDN for the Web App
echo http://$webappname.azurewebsites.net
```



# Creating a Web App with Azure PowerShell

| Command | Notes |
| --- | --- |
| **`New-AzResourceGroup`** | Creates a resource group in which all resources are stored |
| **`New-AzAppServicePlan`** | Creates an App Service plan |
| **`New-AzWebApp`** | Creates an Azure Web App |
| **`Set-AzResource`** | Modifies a resource in a resource group |

# Creating an App Service plan with Azure PowerShell

```
# Create variables for the repository URL, Web App name and location
$gitrepo="https://github.com/Azure-Samples/app-service-web-dotnet-get-started.git"
$webappname="mywebapp$(Get-Random)"
$location="West Europe"

# Create new resource group
New-AzResourceGroup -Name myResourceGroup -Location $location

# Create new App Service plan
New-AzAppServicePlan -Name $webappname -Location $location -ResourceGroupName
myResourceGroup -Tier Free
```

# Creating a Web App with Azure PowerShell

```
# Create new Web App
New-AzWebApp -Name $webappname -Location $location -AppServicePlan $webappname -
ResourceGroupName myResourceGroup

# Create deployment resource manually using ARM
$PropertiesObject = @{
    repoUrl = "$gitrepo";
    branch = "master";
    isManualIntegration = "true";
}
Set-AzResource -PropertyObject $PropertiesObject -ResourceGroupName myResourceGroup -
ResourceType Microsoft.Web/sites/sourcecontrols -ResourceName $webappname/web -
ApiVersion 2015-08-01 -Force
```

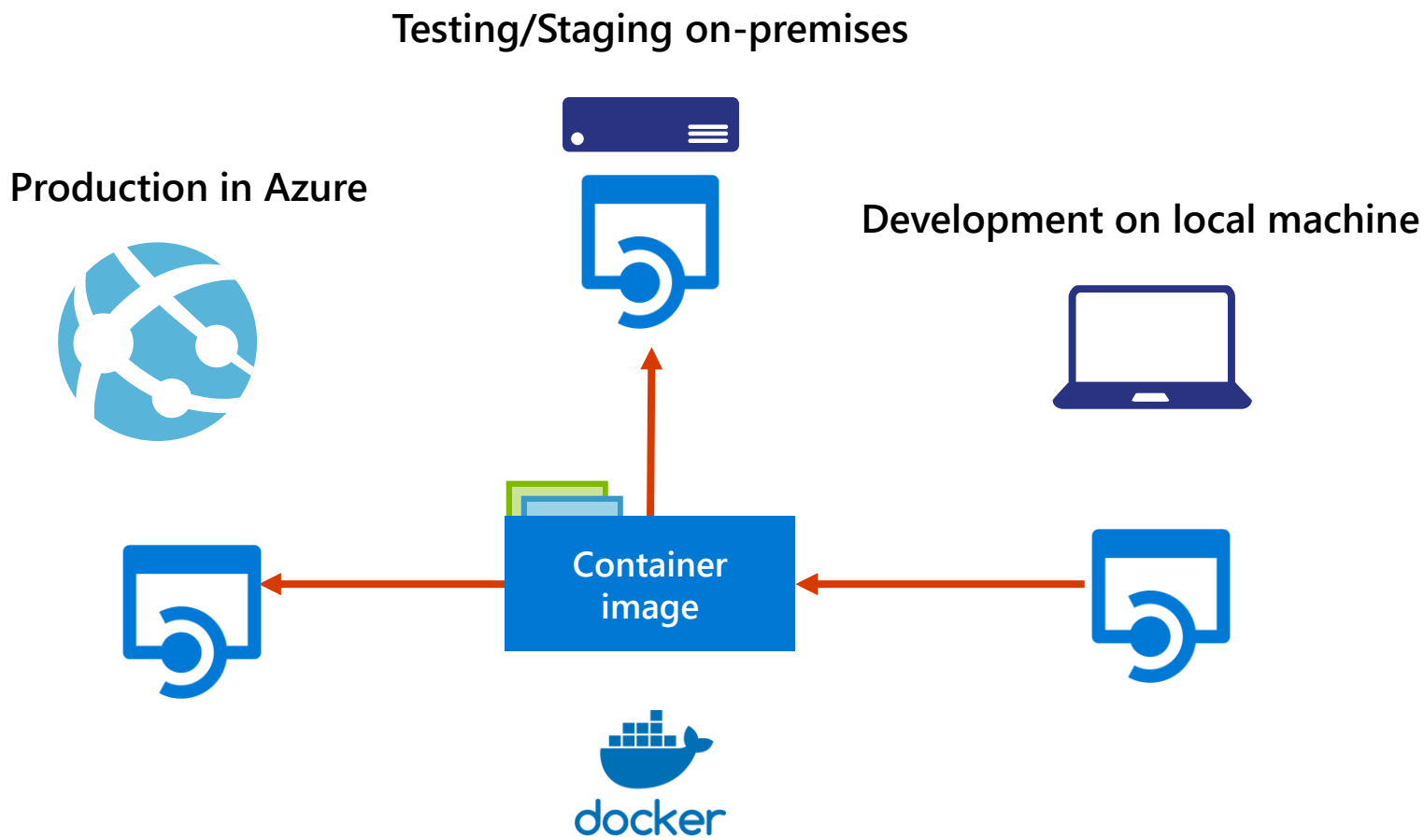
# Demonstration: Creating a static HTML web app by using Azure Cloud Shell

# App Service on Linux

## Why Linux?

- Many application stacks are optimized for Linux:
  - Ruby/Rails, PHP, Node, and others
  - Often, better tools are available on Linux for these stacks
- New and upcoming frameworks are built for Linux first and then Windows
- Portability of Docker containers
- Linux is at the forefront of innovations in nano and microservice architecture

# Docker in App Service on Linux



# Web apps for Linux containers

Deploy applications and solutions that are containerized directly to App Service Web Apps:

- Simplifies deployment
- Matches the already popular container workflow using:
  - CI/CD with Docker Hub, Azure Container Registry, or GitHub
- Compatible with existing App Service features:
  - Auto-scale, Deployment Slots, and others

# Web apps for Linux containers (continued)

Containers can be sourced from your existing registries:

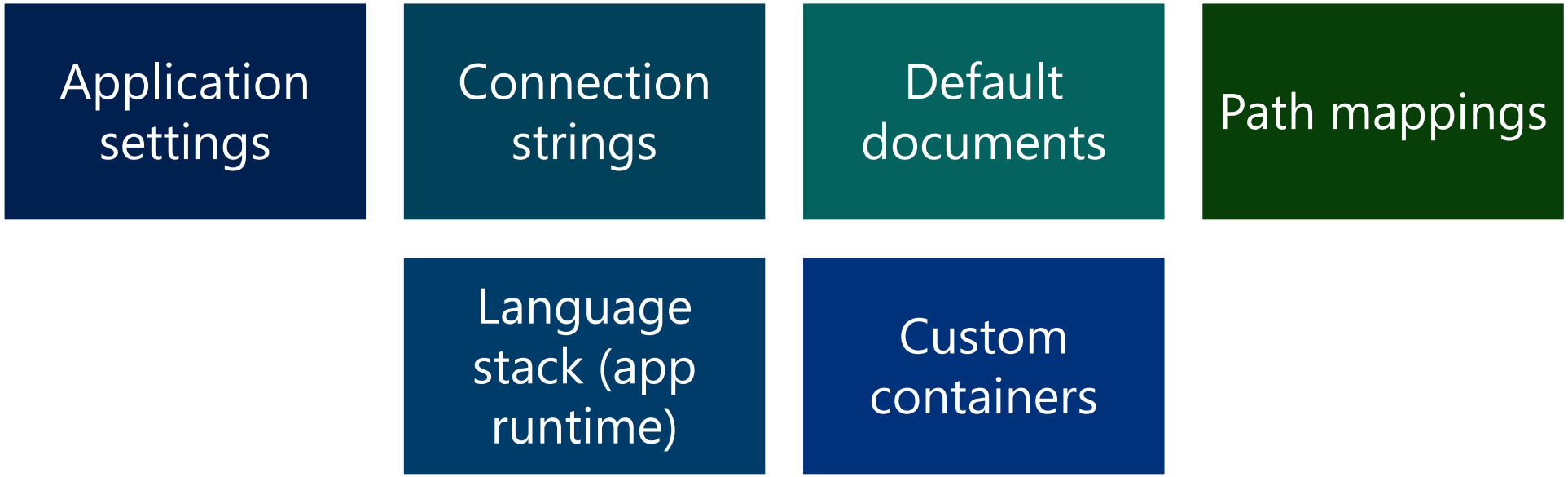
- Docker Hub:
  - Deploy images already shared on Docker Hub
  - Deploy the most popular official images
  - Private images are available on Docker Hub
- Azure Container Registry:
  - Managed service for hosting Docker images
  - Can deploy to Docker Swarm, Kubernetes, or Web App for Containers

# Demonstration: Creating Web Apps with a local Git deployment source

# Lesson 03: Configuring and Monitoring App Service apps

# App Service settings

- Overrides settings in Web.config or appsettings.json
- Hidden by default in Azure portal
- You can configure:



# Default documents

- Only available for Windows apps
- List of documents to show when navigating to a directory on the web server:
  - First matching file is used
- Alternative to building a custom module

# Path mappings

- Windows:
  - Custom IIS handler mappings
  - Virtual applications/directories
- Containerized:
  - Custom-mounted storage

# Updating app runtimes

```
az webapp config set --linux-fx-version "DOTNETCORE|3.1" --resource-group <groupname> --name <appname>
```

Update .NET version

```
az webapp config set --php-version 7.0 --resource-group <groupname> --name <appname>
```

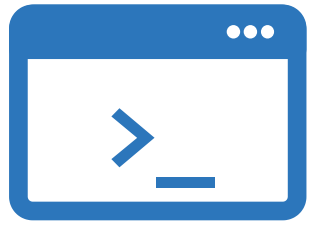
Update PHP version

```
az webapp config set --python-version 3.4 --resource-group <groupname> --name <appname>
```

Update Python version

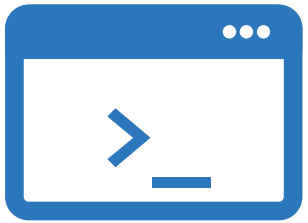
```
az webapp config set --resource-group <groupname> --name <appname> --java-version 1.8 --java-container Tomcat --java-container-version 9.0
```

Update Java version



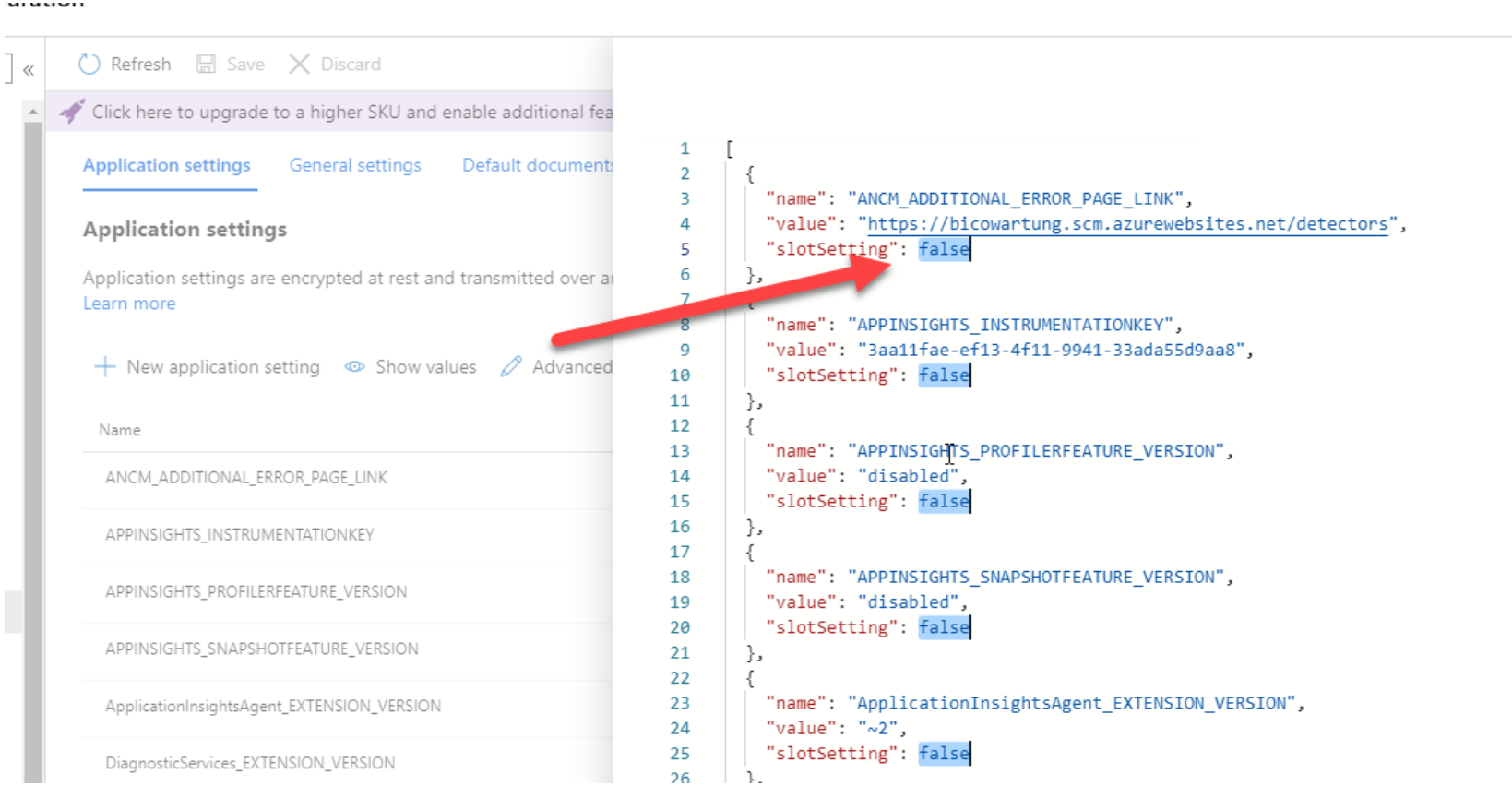
# Updating app runtimes (Node.js)

```
az webapp config appsettings set `
    --resource-group <groupname> `
    --name <appname> `
    --settings WEBSITE_NODE_DEFAULT_VERSION=8.9.3
```



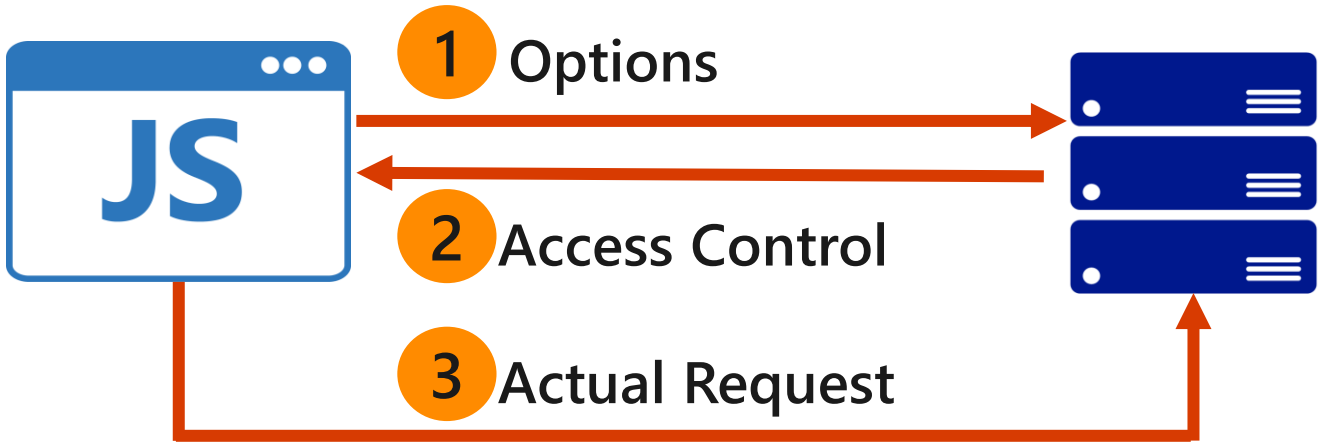
# Editing bulk settings

- Allows editing App Settigs in Bulk



# CORS

- Mechanism for servers to indicate that they support cross-site requests
  - Servers can specify:
    - Allowed HTTP verbs
    - Allowed origins
    - Allowed headers
- Directly supported by API Apps



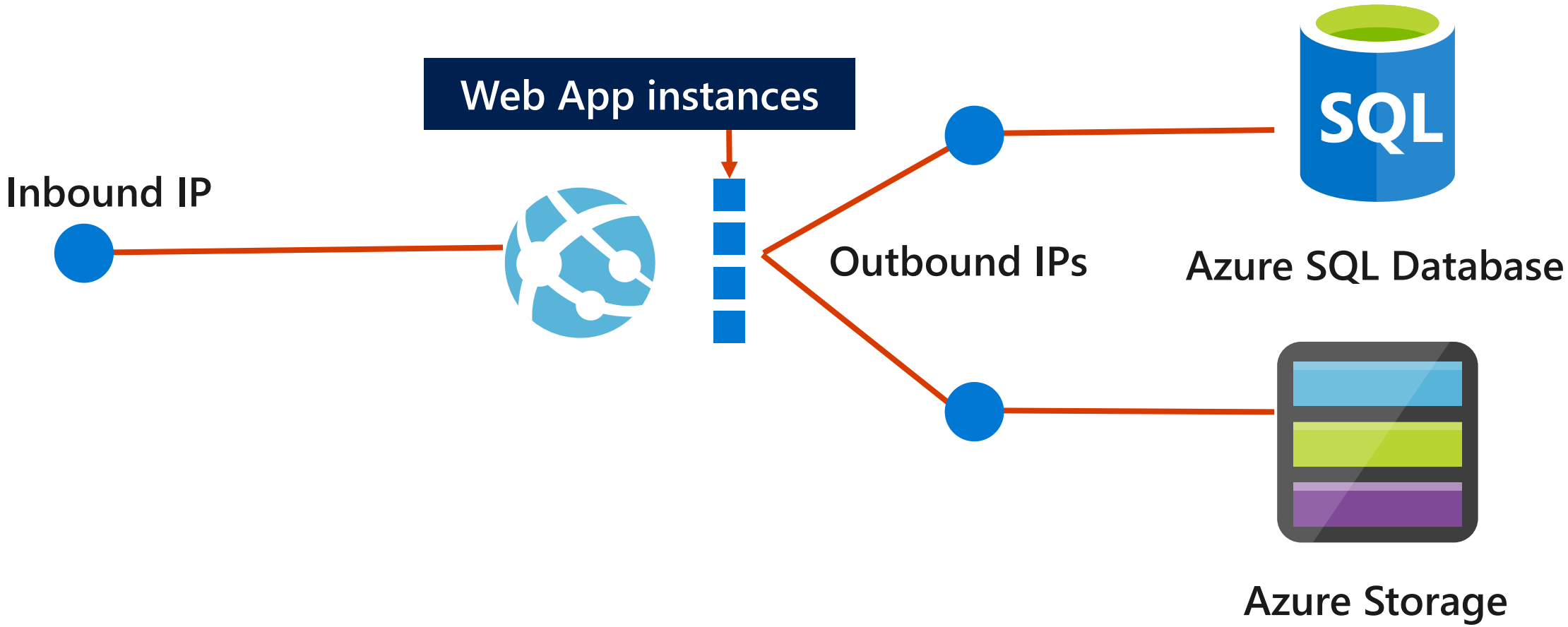
# OS and runtime patching

- OS and application stack are managed by Azure on your behalf
- Monthly OS patching:
  - Physical servers
  - Guest virtual machines
- Stable versions of application runtimes are periodically added to App Services:
  - Some are installed side by side, while others replace existing versions
  - You can manually migrate from one application runtime to another

# Inbound and outbound IP addresses

- Each app has a single inbound IP address:
  - Regardless of scale-out quantity
- Inbound IP address can change:
  - Delete an app and re-create it in a new resource group
  - Delete the last app in a resource group + region combination and re-create it
  - Delete an existing SSL binding
- You can opt to use a static inbound IP
- Each app has a set number of outbound IP addresses:
  - The set and quantity changes as you scale your app between tiers

# Outbound IP addresses

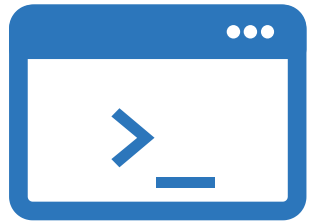


# When IP addresses change

- Inbound:
  - When you delete an app and recreate it
  - When you delete the last app in a resource group
  - When you delete an existing SSL binding
- Outbound:
  - Why you scale from a lower tier (Basic, Standard, Premium) to the Premium V2 tier

# Find outbound IP addresses

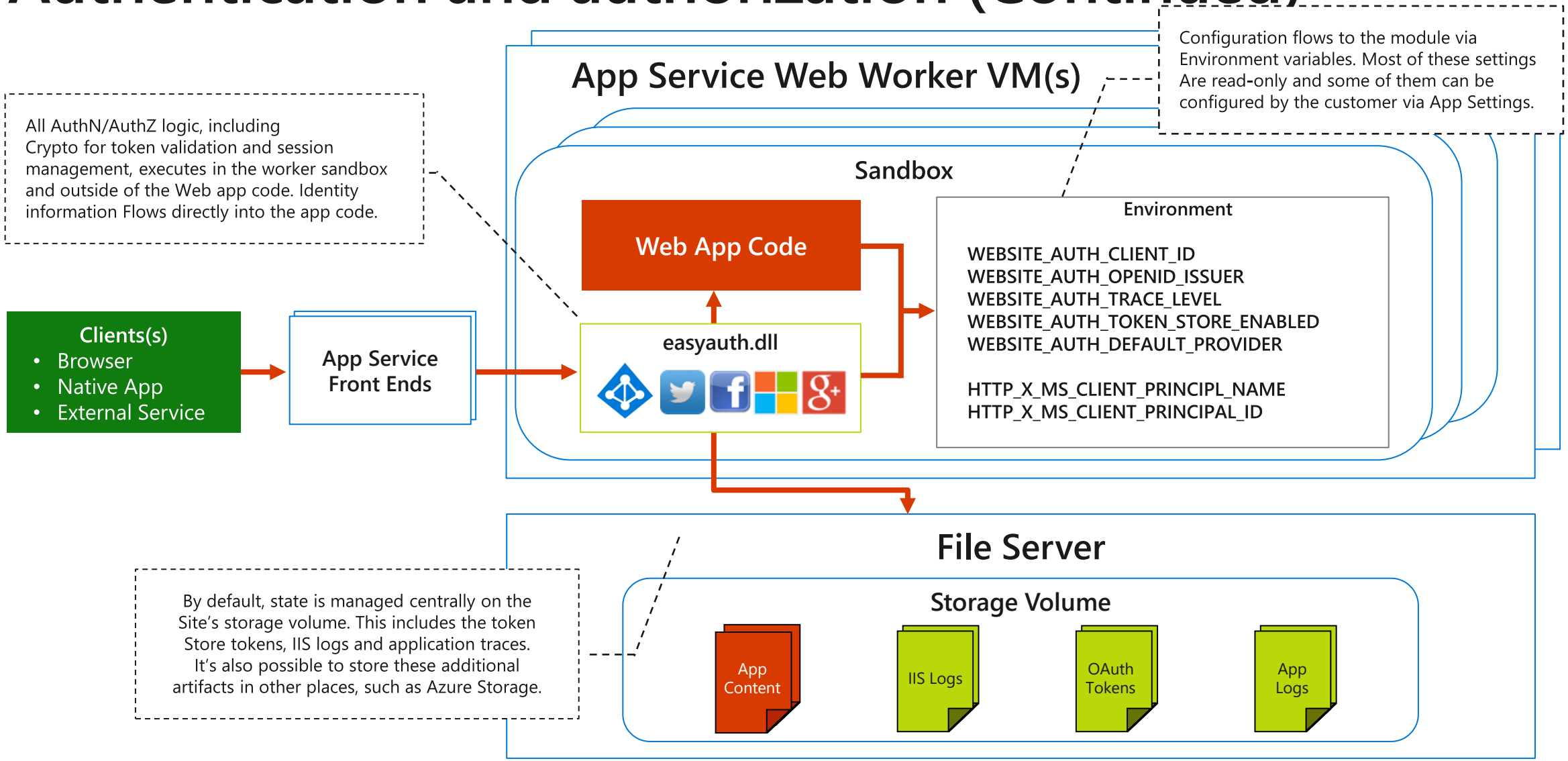
```
# find Outbound IP address
az webapp show --resource-group <group_name> --name <app_name> --query
outboundIpAddresses --output tsv

# find all possible IP addresses (regardless of tier)
az webapp show --resource-group <group_name> --name <app_name> --query
possibleOutboundIpAddresses --output tsv
```



# Authentication and authorization

- Built-in authentication and authorization support:
  - No extra code required to make use of these features
- User claims are made available to code:
  - If you wish to enhance the authentication support, you can use your existing code with popular identity frameworks:
    - ASP.NET Identity
    - PHP server variables
- Built-in token store
- Logging and tracing enabled for authentication events
- Support for popular identity providers:
  - Azure Active Directory (Azure AD), Microsoft accounts, Facebook, Google, Twitter, more…

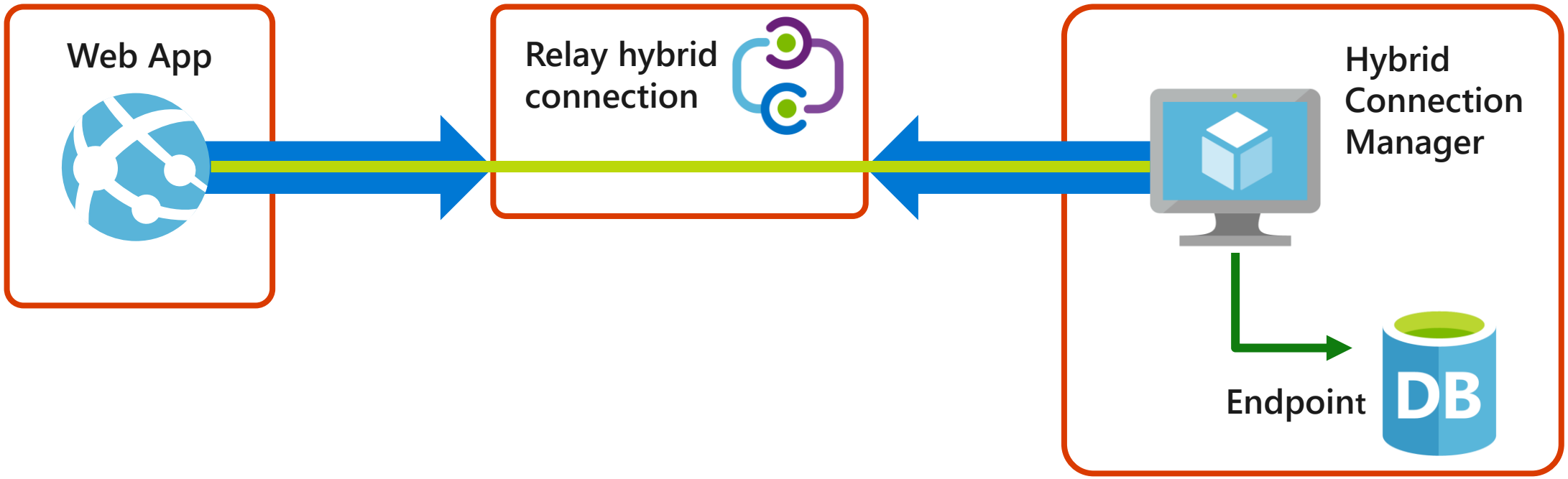
# Authentication and authorization (Continued)



# Azure App Service Hybrid Connections

- Enables access to resources in other networks:
  - Any operating system and any application
  - Hosted in other cloud networks, local networks, or even a specific machine
- Correlates to a single TCP host and port combination
- Service that is available in more than just App Service
- Benefits:
  - Doesn’t require internet-facing endpoint
  - Web Apps can access on-premises systems securely
  - Does not require firewall changes in most scenarios
  - Framework and operating system agnostic

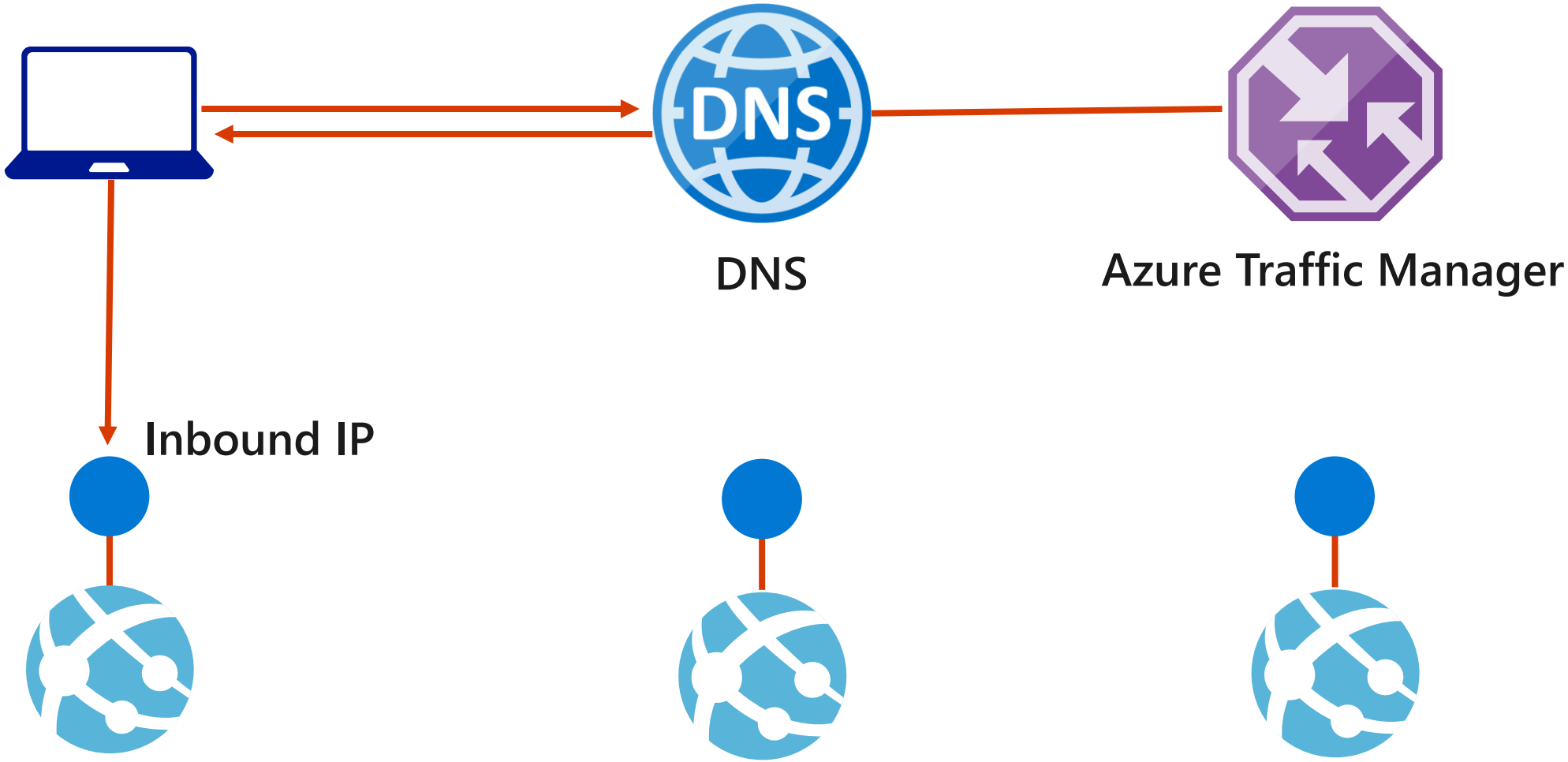
# Azure App Service Hybrid Connections example



# Controlling traffic by using Azure Traffic Manager

- Routes requests from clients to apps in Azure
- Keeps track of app status (running, stopped, deleted):
  - Will automatically route traffic away from an unavailable app
- Configured by using profiles:
  - Stores the routing method for requests
  - Stores a list of endpoints (apps) to route requests to
  - Stores information about endpoint status

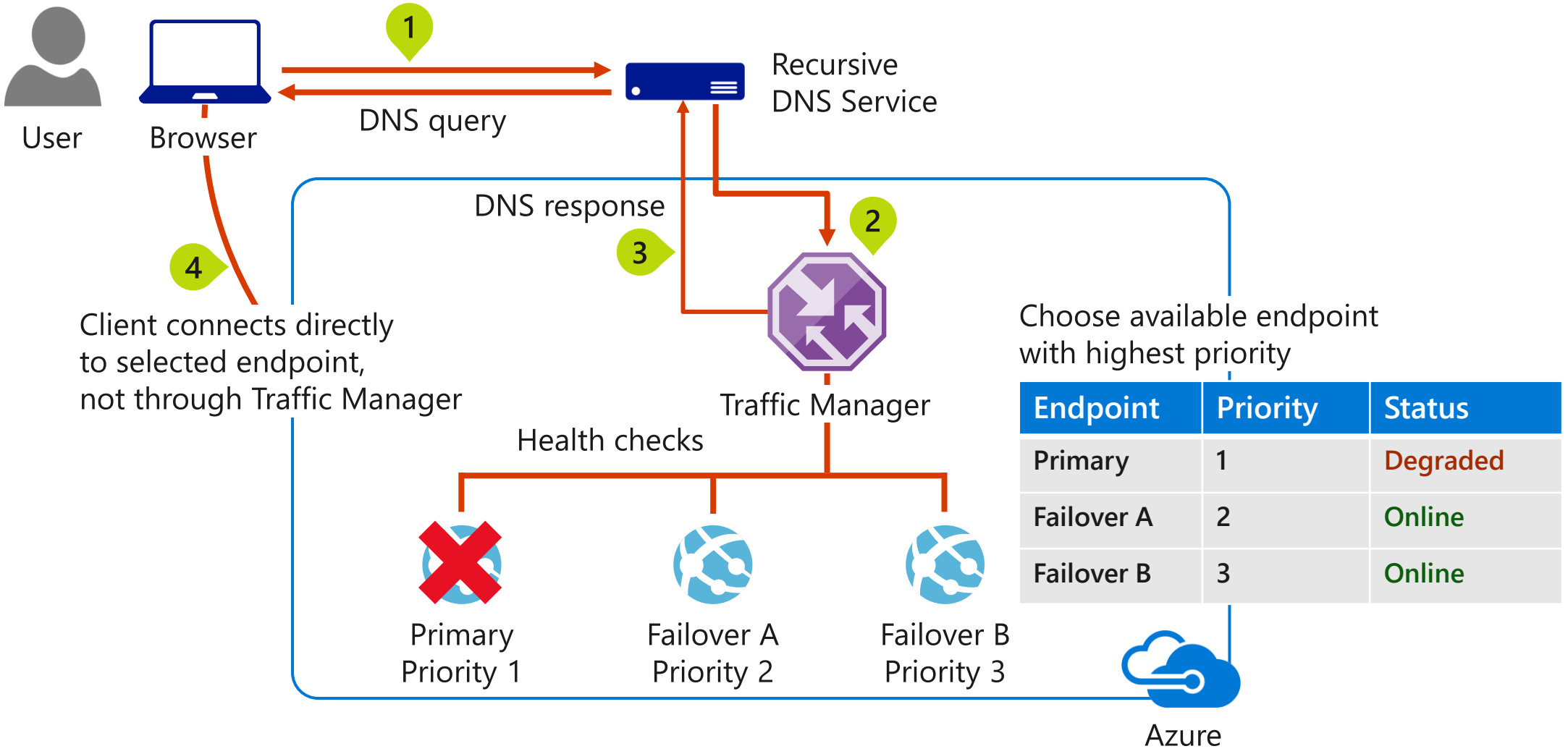
# Azure Traffic Manager and Web Apps



# Azure Traffic Manager routing methods

- Priority:
  - Distribute users to a specific app
  - In case of failure, route users to backup apps based on a priority scheme
- Weighted:
  - Distribute traffic across apps according to weights that you define
  - Your weight definition could potentially distribute users evenly
- Performance:
  - Route users to the “closest” app location based on latency
- Geographic:
  - Route users to specific app locations based on their current location

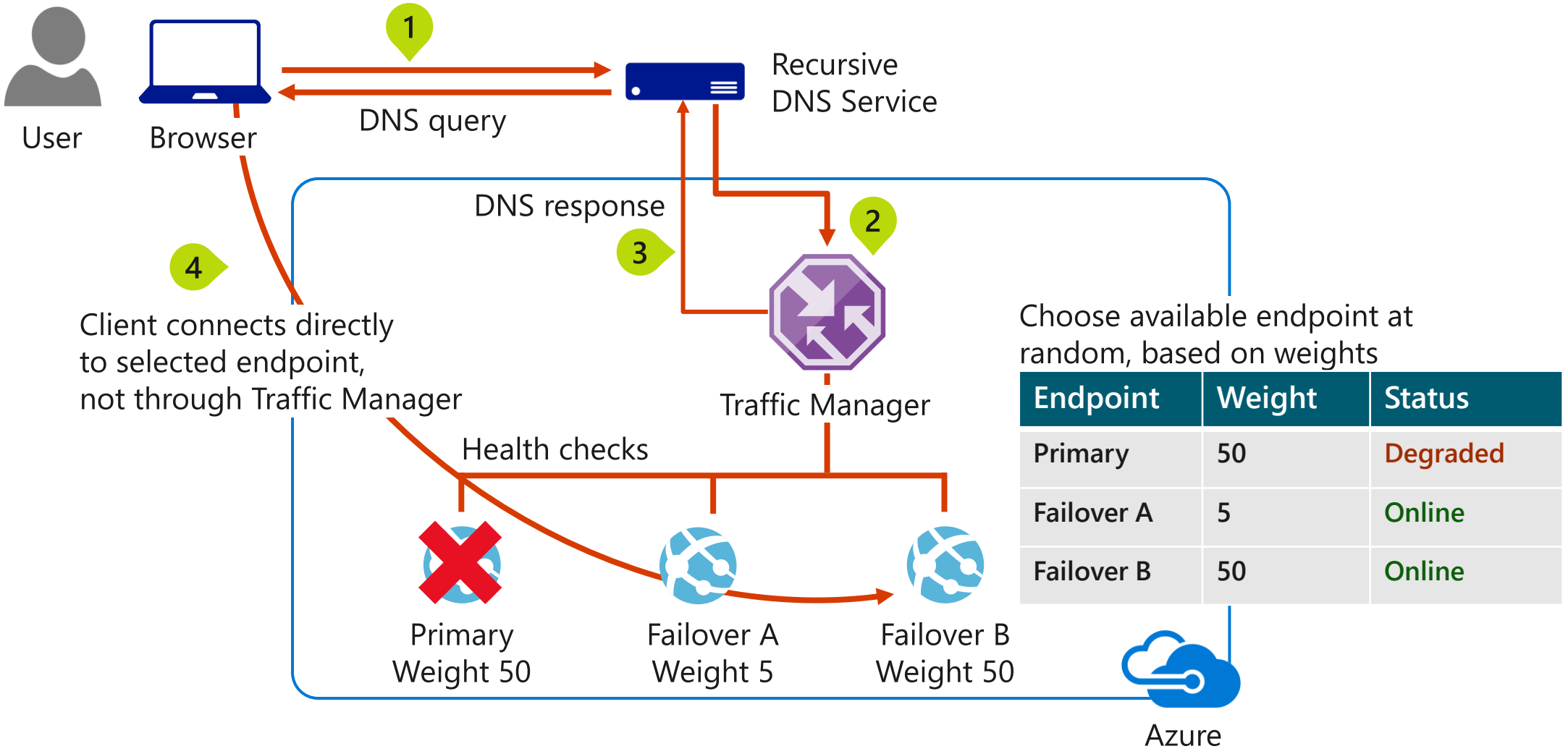
# Priority traffic-routing method




Choose available endpoint with highest priority

| Endpoint | Priority | Status |
|---|---|---|
| Primary | 1 | Degraded |
| Failover A | 2 | Online |
| Failover B | 3 | Online |

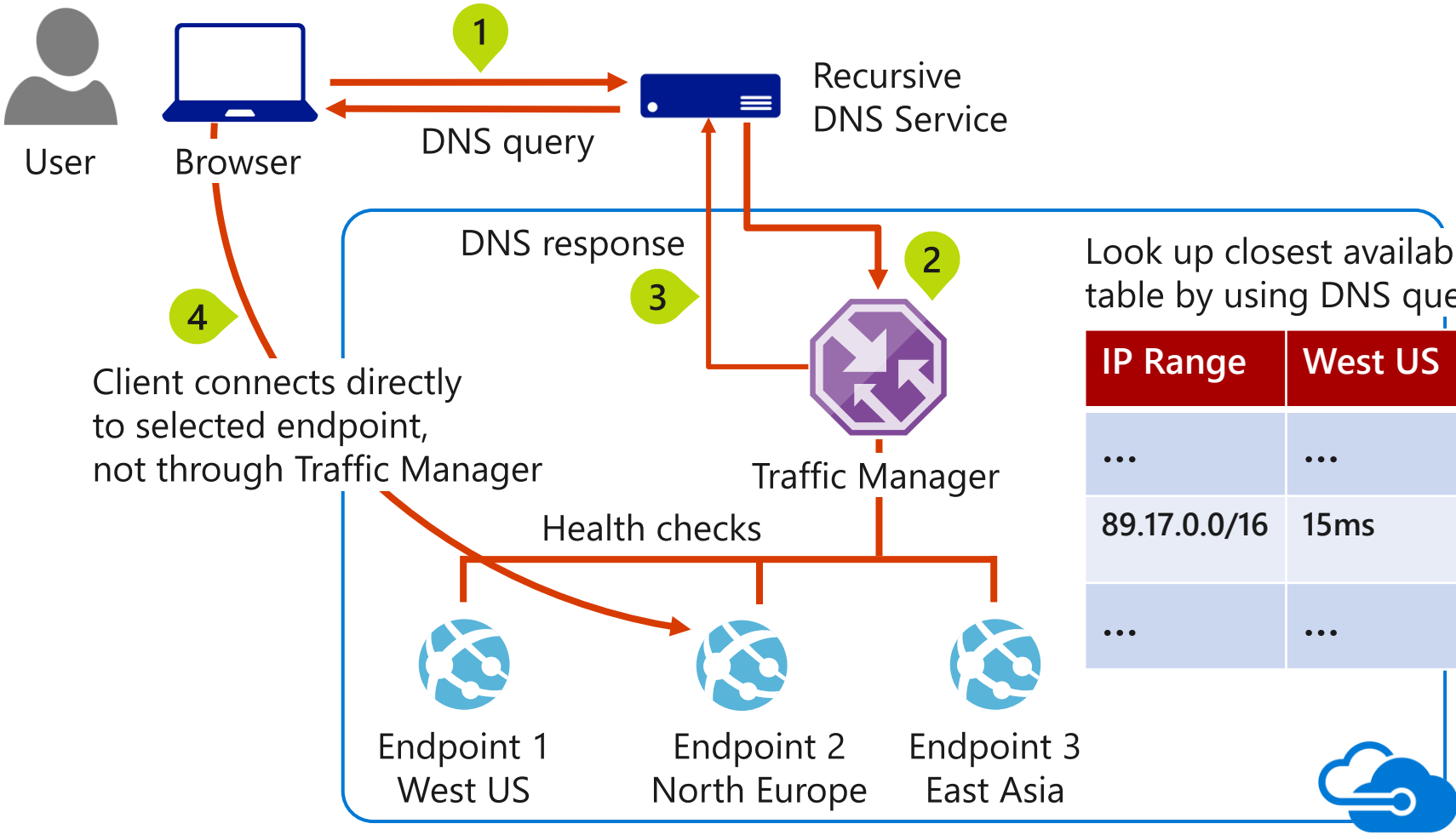
# Weighted traffic-routing method




Choose available endpoint at random, based on weights

| Endpoint | Weight | Status |
|---|---|---|
| Primary | 50 | Degraded |
| Failover A | 5 | Online |
| Failover B | 50 | Online |

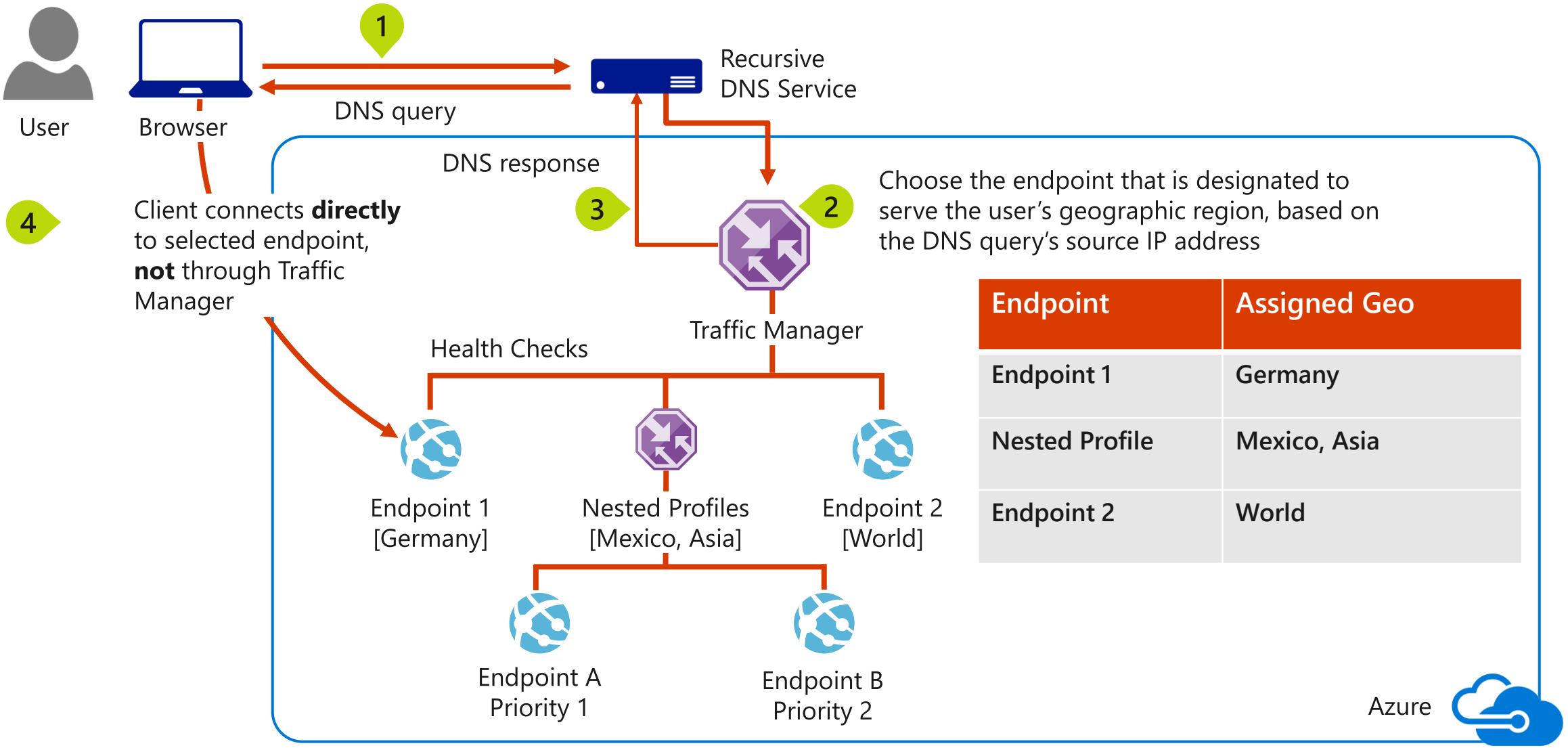
# Performance traffic-routing method




Look up closest available endpoint in latency table by using DNS query source IP address

| IP Range | West US | North Europe | East Asia |
|---|---|---|---|
| … | … | … | … |
| 89.17.0.0/16 | 15ms | 75ms | 150ms |
| … | … | … | … |

# Geographic traffic-routing method




| Endpoint | Assigned Geo |
|---|---|
| Endpoint 1 | Germany |
| Nested Profile | Mexico, Asia |
| Endpoint 2 | World |

# Azure App Service Local Cache

- Provides a write-but-discard cache of your content
- Created asynchronously when site is started
- Automatically serves content once the cache is ready
- Faster than reading content from Azure Storage directly on each client request

# App Service Environments (ASEs)

- App Service variant that provides a fully isolated and dedicated environment for securely running App Service apps at high scale
- Ideal for application workloads that require:
  - Very high scale, higher than typical App Service capacity
  - Network isolation and secure network access
  - High memory utilization
- Single or Multi-region
- Deployed to a virtual network
- An ASE is dedicated exclusively to a single subscription:
  - Max 100 instances

# Lesson 04: Scaling App Service apps

# Autoscale

- A primary advantage of the cloud is elastic scaling (the ability to use as much capacity as you need):
  - Scaling out as load increases
  - Scaling in when the extra capacity is not needed
- Many Microsoft Azure services provide the capability to scale both manually and automatically
- Autoscale refers to the capability of many of these services to monitor the application instances and automatically scale appropriately to handle the current usage of the application:
  - Using autoscale, your cloud service can scale out and in to exactly match the amount of instances needed for your specific computing pattern

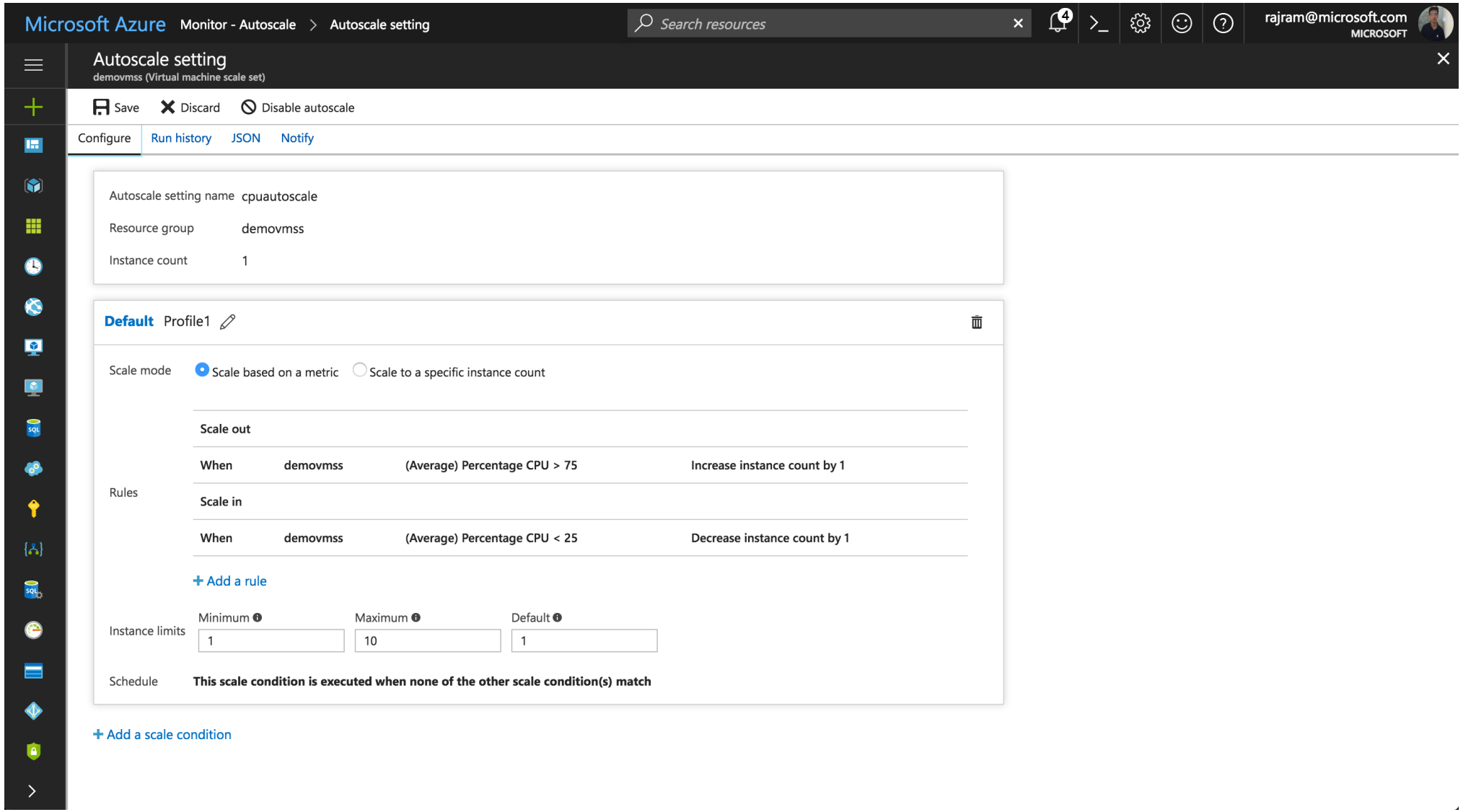
# Autoscale metrics

| Metric | Metric identifier | Description |
|---|---|---|
| **CPU** | CpuPercentage | The average amount of CPU time used across all instances of the plan |
| **Memory** | MemoryPercentage | The average amount of memory used across all instances of the plan |
| **Data in** | BytesReceived | The average incoming bandwidth used across all instances of the plan |
| **Data out** | BytesSent | The average outgoing bandwidth used across all instances of the plan |
| **HTTP queue** | HttpQueueLength | The average number of both read and write requests that were queued on storage. A high disk queue length is an indication of an application that might be slowing down due to excessive disk I/O. |
| **Disk queue** | DiskQueueLength | The average number of HTTP requests that had to sit in the queue before being fulfilled. A high or increasing HTTP queue length is a symptom of a plan under a heavy load. |

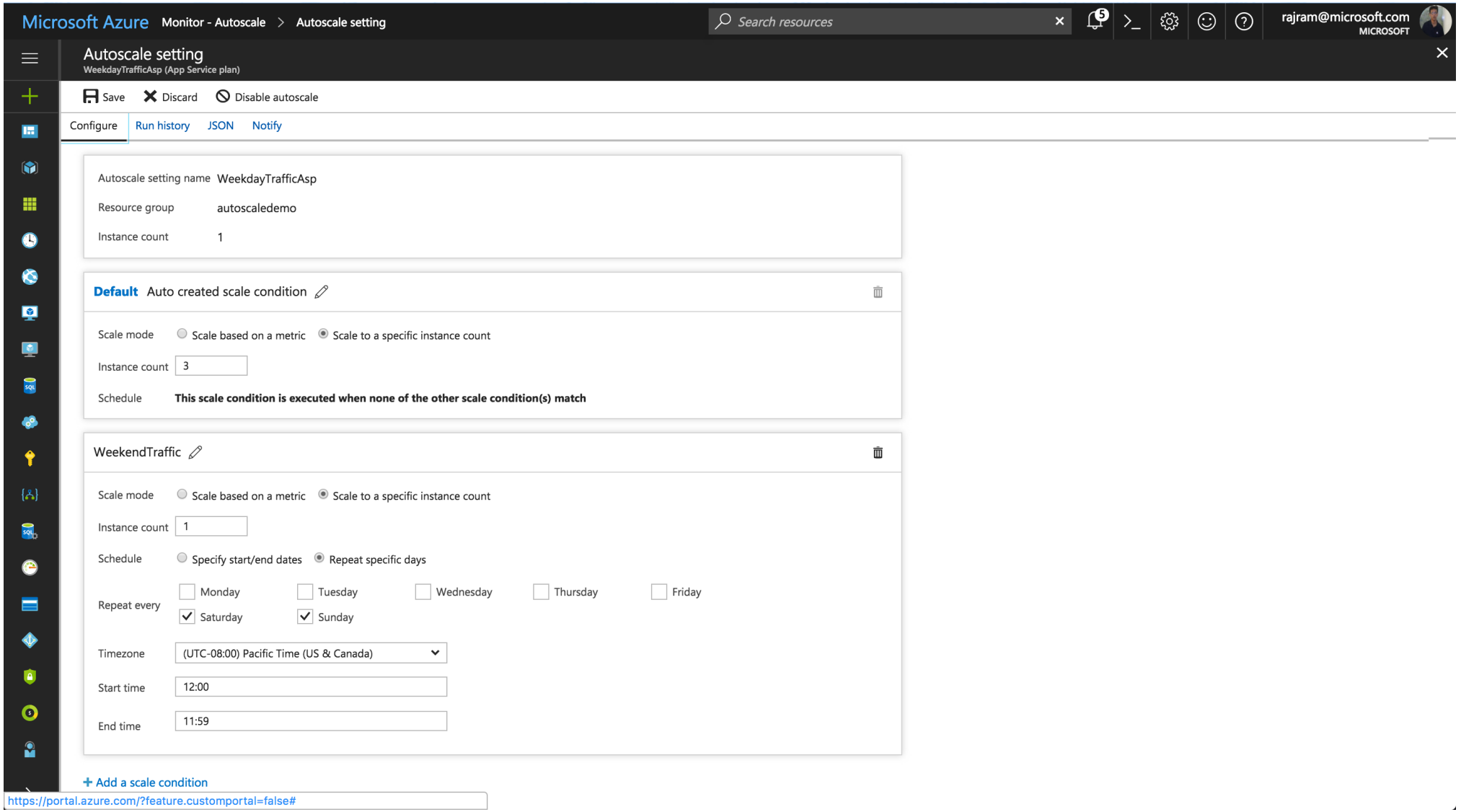
# Autoscale patterns

- Scale based on CPU
- Scale differently on weekdays vs. weekends
- Scale differently during holidays
- Scale based on custom metric

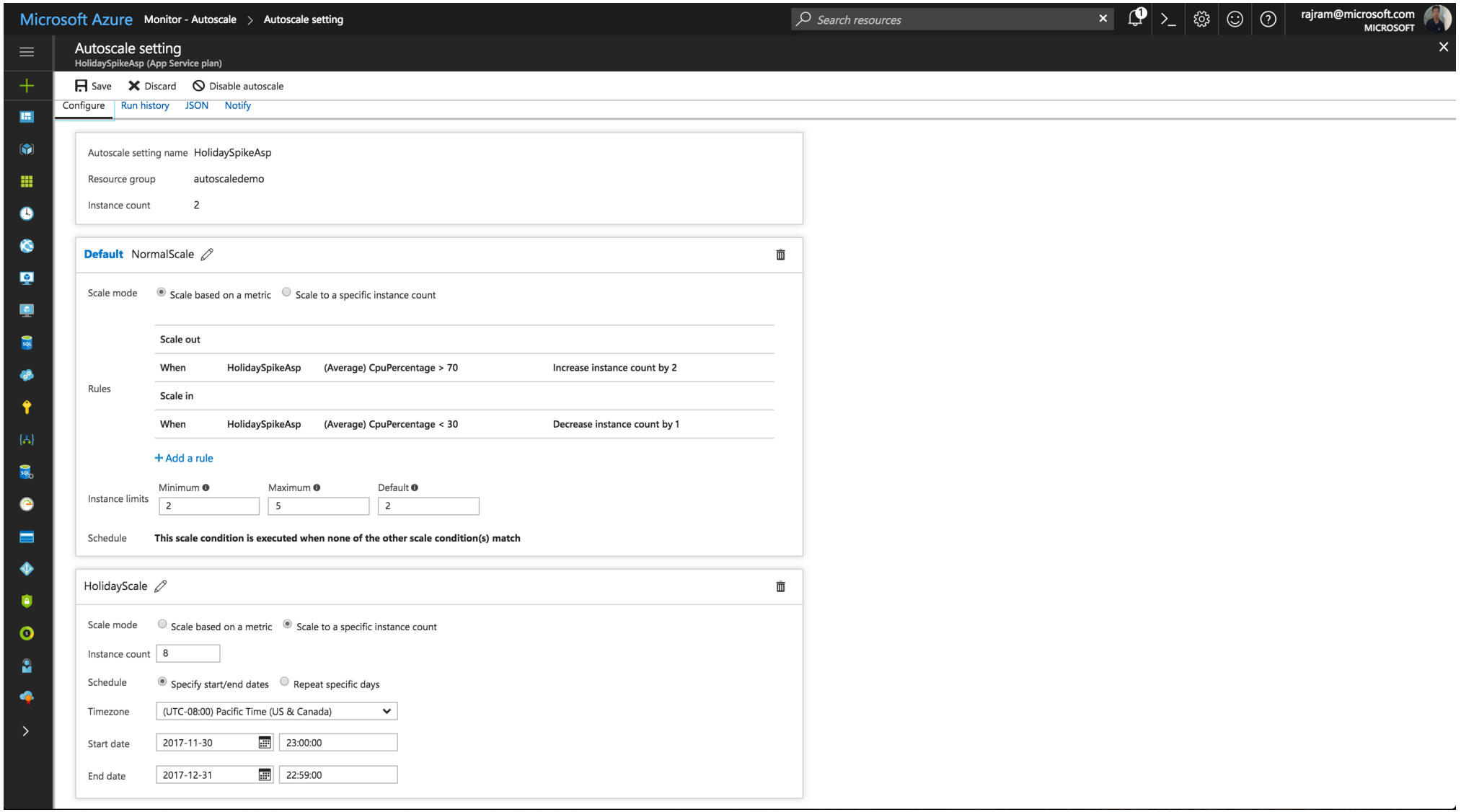
# Scale based on CPU



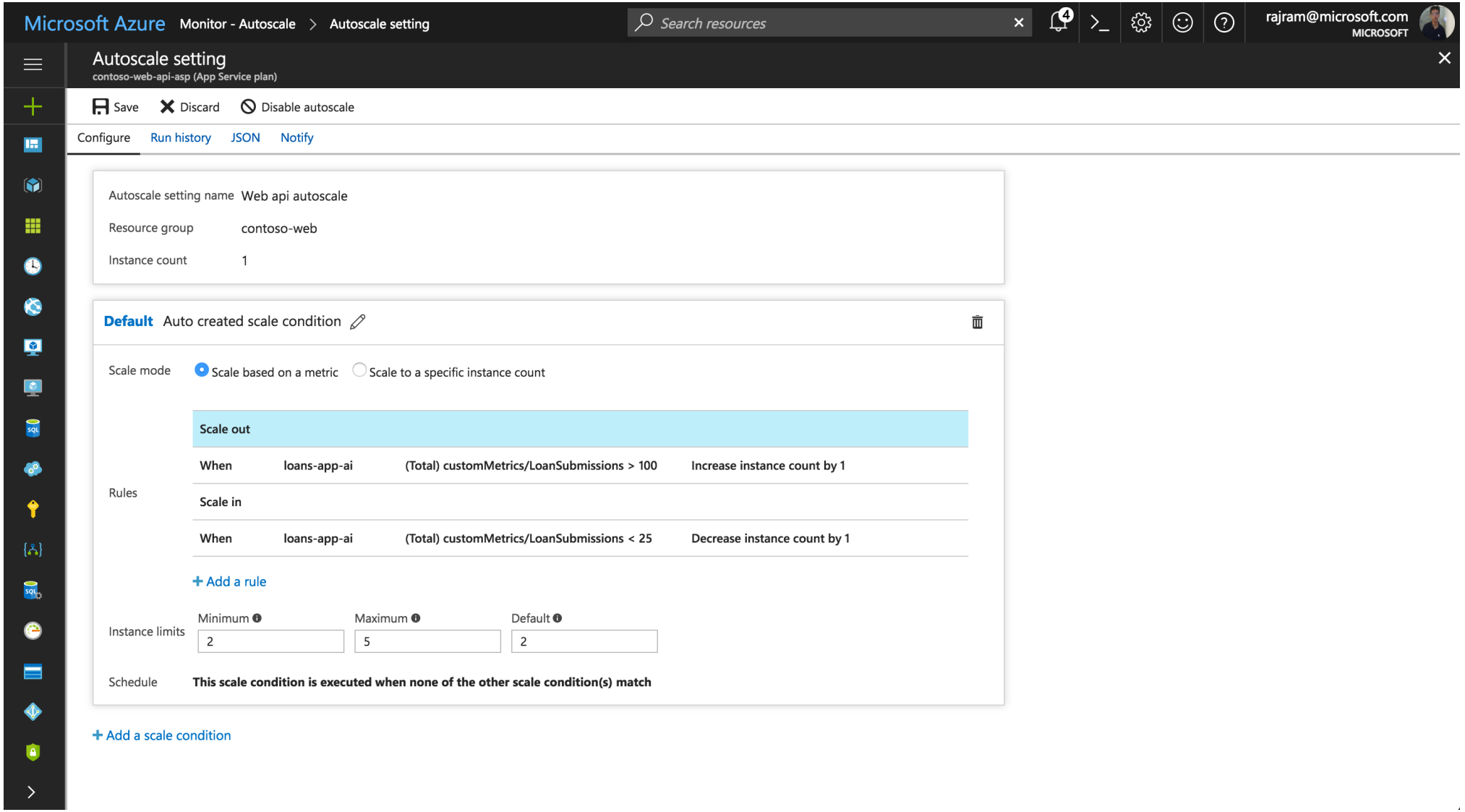
# Scale differently on weekdays vs. weekends



# Scale differently during holidays



# Scale based on custom metric



# Autoscale setting schema

```
{
    "id": "...", "name": "demoSetting", "type": "Microsoft.Insights/autoscaleSettings",
    "location": "East US",
    "properties": {
        "enabled": true, "targetResourceUri": "...",
        "profiles": [
            {
                "name": "mainProfile",
                "capacity": {
                    "minimum": "1", "maximum": "4", "default": "1"
                },
                "rules": [ ... ]
            }
        ]
    }
}
```

# Autoscale setting schema – scale-out rule

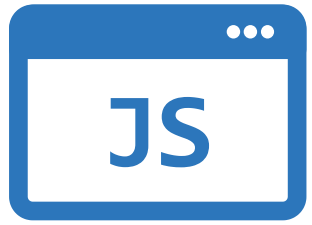
```
{
    "metricTrigger": {
        "metricName": "Percentage CPU", "metricResourceUri": "...",
        "timeGrain": "PT1M",
        "statistic": "Average",
        "timeWindow": "PT10M",
        "timeAggregation": "Average",
        "operator": "GreaterThan",
        "threshold": 85
    },
    "scaleAction": {
        "direction": "Increase", "type": "ChangeCount",
        "value": "1",
        "cooldown": "PT5M"
    }
}
```

# Autoscale setting schema – scale-in rule

```
{
    "metricTrigger": {
        "metricName": "Percentage CPU", "metricResourceUri": "...",
        "timeGrain": "PT1M",
        "statistic": "Average",
        "timeWindow": "PT10M",
        "timeAggregation": "Average",
        "operator": "LessThan",
        "threshold": 60
    },
    "scaleAction": {
        "direction": "Decrease", "type": "ChangeCount",
        "value": "1",
        "cooldown": "PT5M"
    }
}
```

# Autoscale setting schema – fixed date profile

```
"fixedDate": {
    "timeZone": "Pacific Standard Time",
    "start": "2020-12-26T00:00:00",
    "end": "2020-12-26T23:59:00"
}
```



# Autoscale setting schema – weekdays and weekends

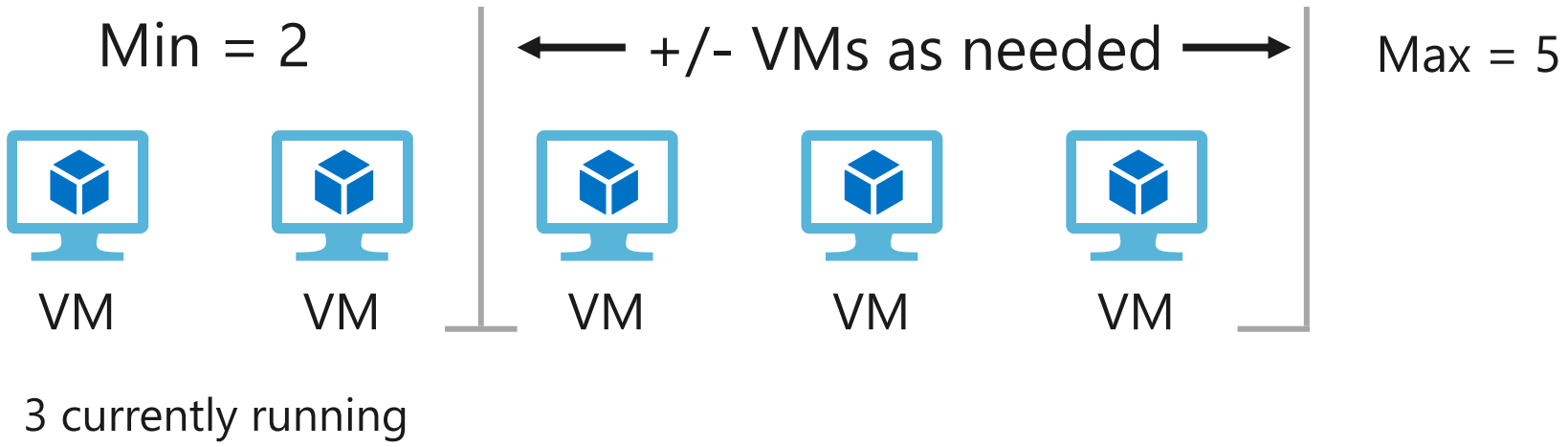
```
"recurrence": {
    "frequency": "Week",
    "schedule": {
        "timeZone": "Pacific Standard Time",
        "days": [
            "Saturday"
        ],
        "hours": [
            0
        ],
        "minutes": [
            0
        ]
    }
}
```

# Autoscale concepts

- Each resource can have one autoscale setting:
  - Autoscale settings can have one-to-many profiles
  - Profiles can have one-to-many rules
- Autoscale increases instances horizontally within bounds:
  - Bounds are set by using the minimum, maximum, and default values
- Thresholds are calculated at an instance level
- Autoscale successful actions and failures are logged to the Activity Log

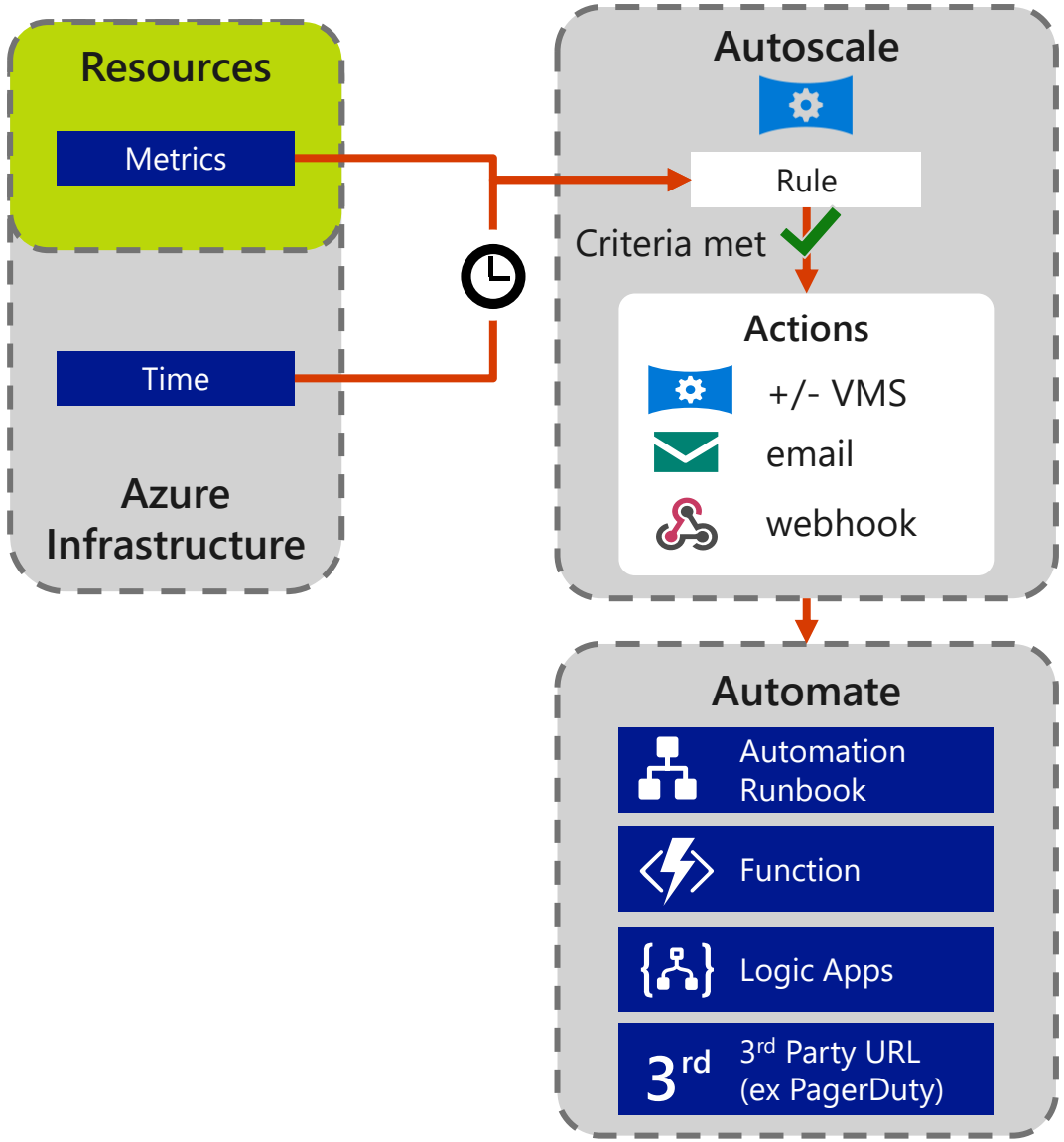
# Autoscale thresholds

- Scale is constrained to a minimum and maximum:
  - Your current instance count must be between the minimum and maximum:
    - Minimum can help guarantee availability
    - Maximum can help control costs



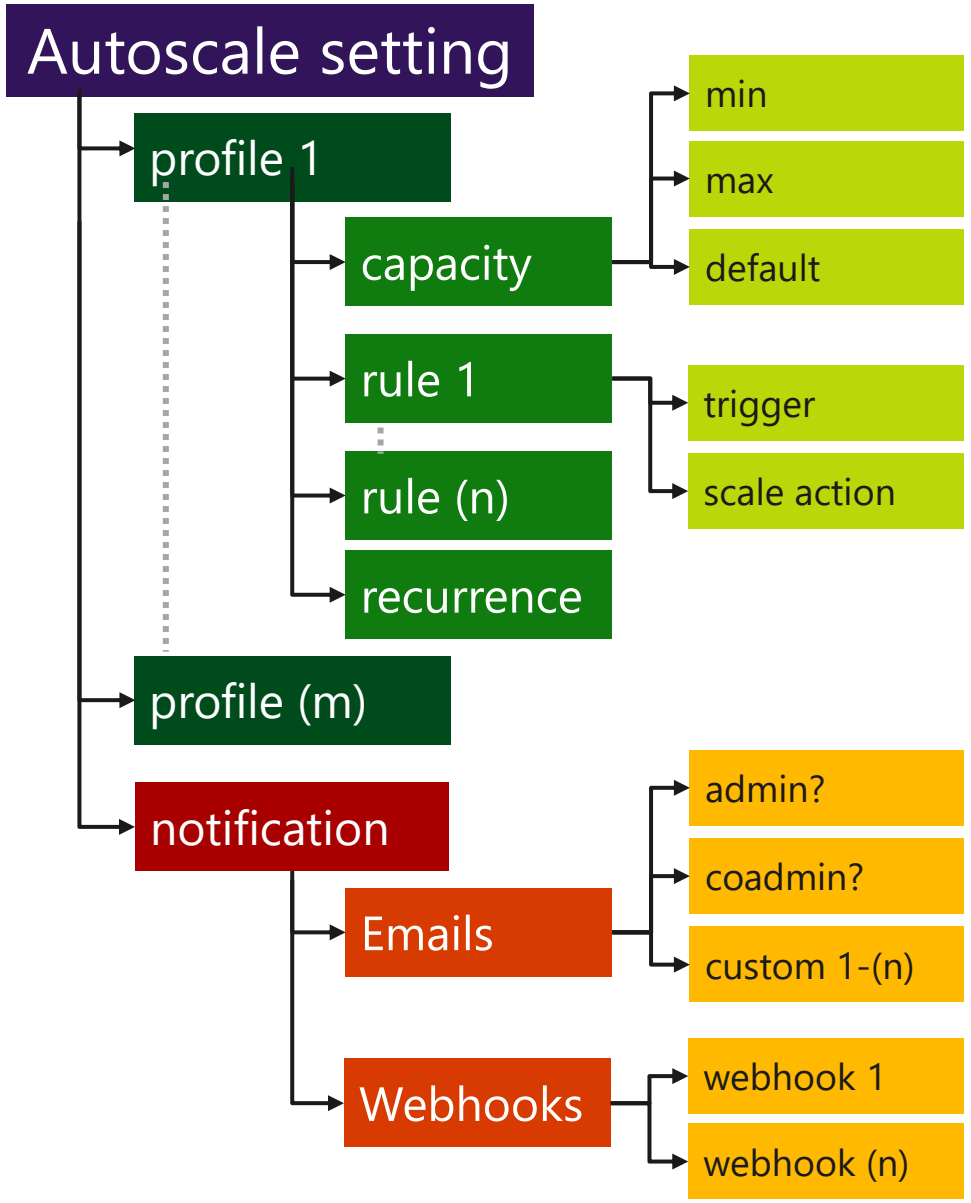
# Autoscale workflow

1. Metrics are measured for a resource
2. When conditions are met (threshold surpassed), autoscale triggers:
   a. Perform scale actions
   b. Send notifications (alerts)
   c. Send messages to webhooks for external automation



# Autoscale hierarchy

- One autoscale setting
- Settings have one or more profiles
- Profiles have one or more rules:
  - Profiles can also have recurrences and capacity settings
- Notifications can be directly associated with an autoscale setting



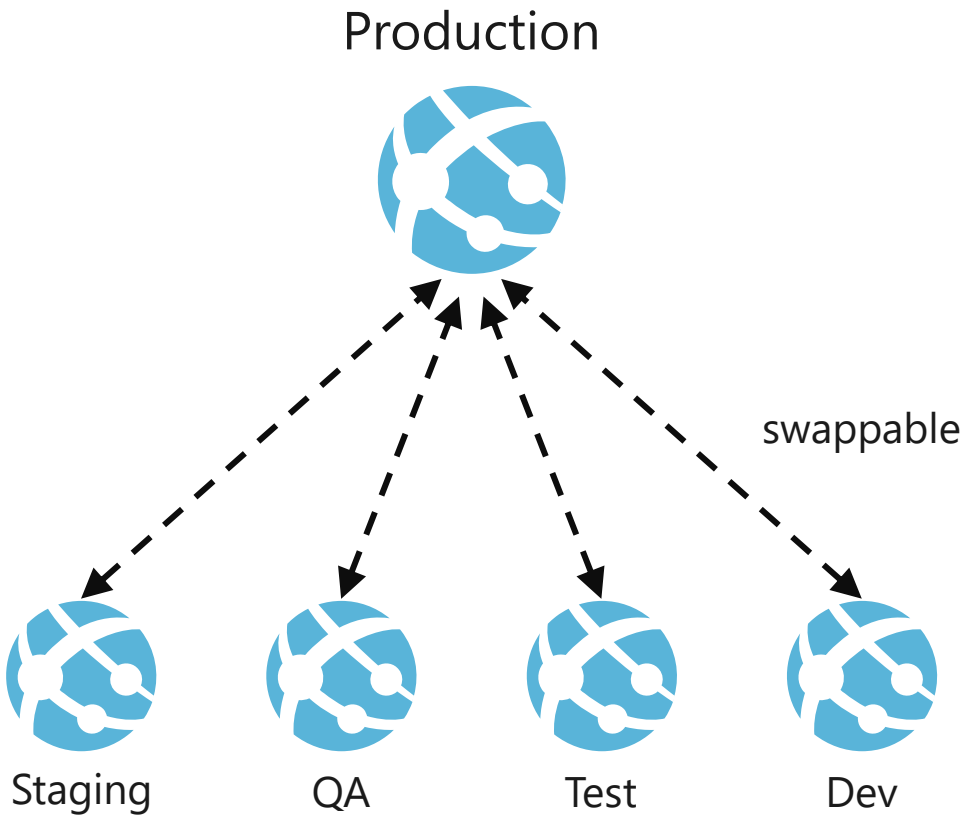
# Best practices

- Ensure that the maximum and minimum values are different and have an adequate margin between them
- Manual scaling is reset by autoscale min and max
- Always use a scale-out and scale-in rule combination that performs an increase and decrease
- Choose the appropriate statistic for your diagnostics metric
- Choose the thresholds carefully for all metric types

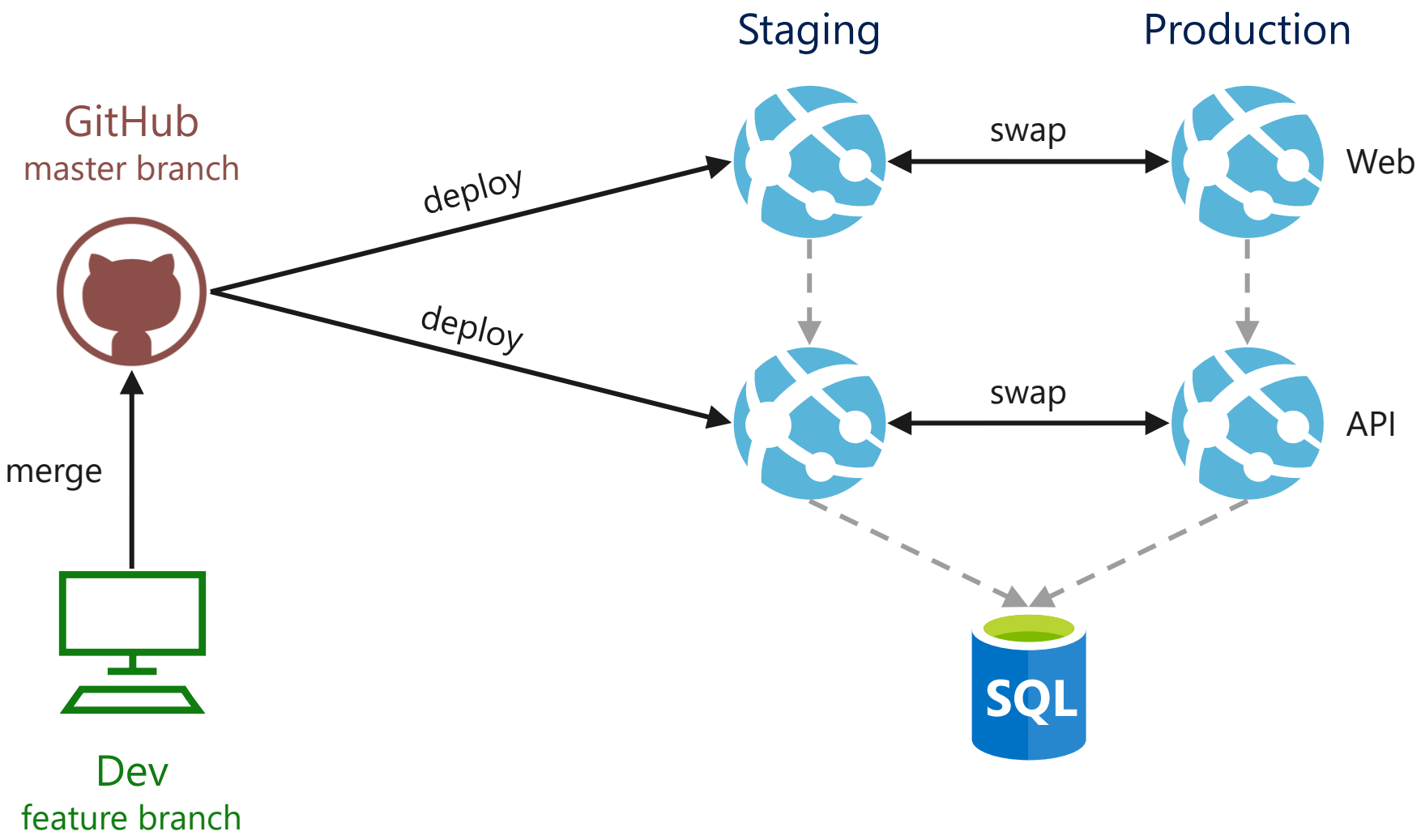
# Lesson 05: Azure App Service staging environments

# Deployment slots

- Live apps with their own:
  - Host names
  - Content
  - Configuration
- Can be swapped between each other.
  - Staging ↔ Production
  - Production ↔ Staging
  - Dev ↔ Test
  - Test ↔ QA
  - QA ↔ Staging



# Modern deployment workflow

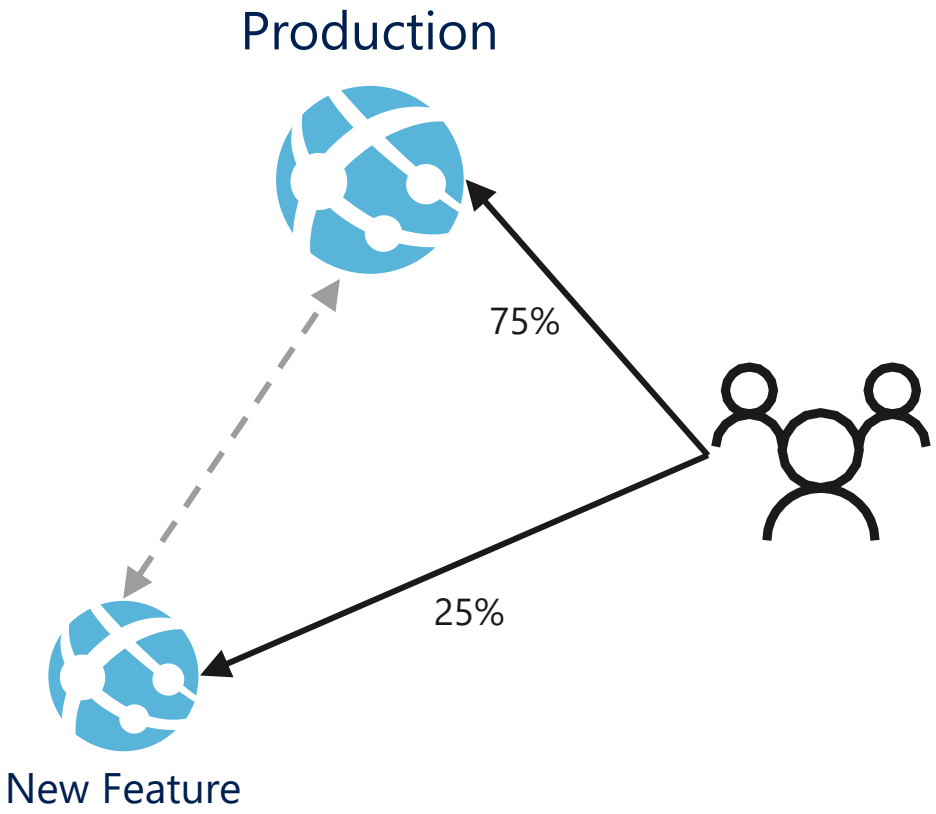


# Auto swap

- Automatically swaps an application to the production slot:
  - Performed after the application is "warmed up"
  - Swaps the deployment target slot with the production slot
- Deploy apps continuously while minimizing cold starts and downtime

# Route traffic between slots

- All traffic is normally routed to production:
  - Production slot has 100% weighting
- You can manually configure the weight of traffic between multiple slots



# Automate slot management - Azure PowerShell

```
New-AzWebAppSlot -ResourceGroupName "<groupname>" -Name "<appname>" -Slot "staging" -
AppServicePlan "<appserviceplanname>"
```

Create new slot

Target

```
$Params = @{targetSlot  = "[slot name – e.g. "production"]"}
```

Source

```
Invoke-AzResourceAction -ResourceGroupName "<groupname>" `
        -ResourceType "Microsoft.Web/sites/slots" -ResourceName "<appname>/staging" `
        -Action "slotsswap" -Parameters $Params -ApiVersion 2015-07-01
```

Swap slots

# Automate slot management - Azure CLI

```
az webapp deployment slot create --name <appname> --resource-group <groupname> --
slot staging
```

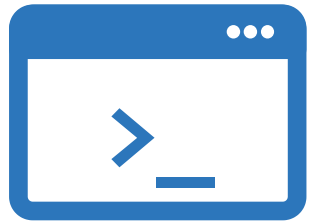
Create new slot

Swap slots

```
az webapp deployment slot swap --resource-group <groupname> --name <appname>
--slot staging --target-slot production
```

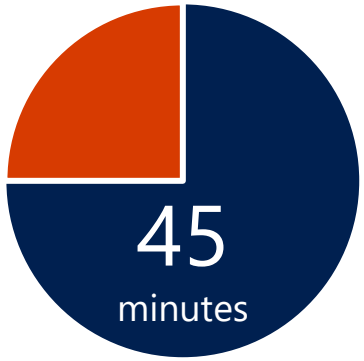
Source

Target



# Lab: Building a web application on Azure platform as a service offerings

## Duration




## Lab sign-in information