

# Introduction

2 minutes

It's important to be able to scale a web app for these reasons:

- It enables the app to remain responsive during periods of high demand.
- It helps to save you money by reducing the resources required when demand drops.

Azure App Service enables you to meet these goals by providing scale up and down, and scale in and out.

Imagine that you work for a large chain of hotels. You have a website that customers can visit to make bookings and to view the details of bookings that they've previously made. At certain times of the year, the volume of traffic grows because customers are browsing hotels for summer vacations. At other times, traffic declines. These patterns are predictable.

In this module, you'll use Azure App Service to scale a web app to match planned seasonal throughput requirements and also meet demand during short-term peak events. This module also describes how to scale up a web app onto more powerful hardware to meet future requirements.

## Learning objectives

In this module, you will:

- Scale a web app in and out manually.
- Scale a web app up and down.

## Prerequisites

- Navigate the Azure portal.
- Use the Azure portal to create a new App Service web app.

**Next unit: Scale a web app manually**

Continue

# Scale a web app manually

7 minutes

By manually scaling out and back in again, you can respond to expected increases and decreases in traffic. Scaling out has the additional benefit of increasing availability because of the increased number of instances of the web app. A failure of one instance doesn't make the web app unavailable.

In the hotel reservation system, you can scale out before an anticipated seasonal influx. You can scale back in when the season is over and the number of booking requests is reduced.

In this unit, you'll learn how to manually scale out a web app and how to scale it back in.


## App Service plans and scalability

A web app running in Azure typically uses Azure App Service to provide the hosting environment. App Service can arrange for multiple instances of the web app to run and will load balance incoming requests across these instances. Each instance runs on a virtual machine.

The resources available to each instance are defined by an App Service plan. The App Service plan specifies the operating system (Windows or Linux), the hardware (memory, CPU processing capacity, disk storage, and so on), and the availability of services like automatic backup and restore.

Azure provides a series of well-defined App Service plan tiers. This list summarizes each of these tiers, in increasing order of capacity (and cost):

- The Free tier provides 1 GB of disk space and support for up to 10 apps, but only a single shared instance and no SLA for availability. Each app has a compute quota of 60 minutes per day. The Free service plan is mainly suitable for app development and testing rather than production deployments.
- The Shared tier provides support for more apps (up to 100) also running on a single shared instance. Apps have a compute quota of 240 minutes per day. There is no availability SLA.
- The Basic tier supports an unlimited number of apps and provides more disk space. Apps can be scaled out to three dedicated instances. This tier provides an SLA of 99.95% availability. There are three levels in this tier that offer varying amounts of compute power, memory, and disk storage.
- The Standard tier also supports an unlimited number of apps. This tier can scale to 10 dedicated instances and has an availability SLA of 99.95%. Like the Basic tier, this tier has three levels that offer an increasingly powerful set of compute, memory, and disk options.
- The Premium tier gives you up to 20 dedicated instances, an availability SLA of 99.95%, and multiple levels of hardware.
- The Isolated tier runs in a dedicated Azure virtual network, which gives you network and compute isolation. This tier can scale out to 100 instances and has an availability SLA of 99.95%.

 **Note**

Some tiers aren't available for all operating systems. For example, there is currently no Shared tier for Linux.

## Monitor and scale a web app

When you create a web app, you can either create a new App Service plan or use an existing one. If you select an existing plan, any other web apps that use the same plan will share resources with your web app. They'll all scale together, so they need to have the same scaling requirements. If your apps have different requirements, use a separate App Service plan for each one.

You scale out by adding more instances to an App Service plan, up to the limit available for your selected tier. If you're not using the Free tier, you're charged for each instance by the hour. You can perform this task in the Azure portal.

The key to scaling effectively is knowing when to scale, and by how much. You monitor the performance of a web app by using the metrics available for the App Service. The simplest way to do this is to use the Azure portal. If you notice a steady increase in resource use, such as CPU utilization, memory occupancy, or disk queue length, you should consider scaling out before these metrics hit a critical point. You should also monitor the average response time of requests and the number of failing requests. If both of these figures are high, the system might be running close to (or beyond) capacity. You might need to scale out immediately.

If the metrics indicate that your system is lightly loaded and has plenty of spare capacity, you might want to scale back in to reduce costs.

In both cases, you should continue to monitor the statistics for the web app. Allow the system to stabilize. If the metrics indicate that the app is still underpowered (or overpowered), add or remove instances as needed.

# Exercise - Scale a web app manually

15 minutes

You should scale out a system when you expect an increase in traffic. You might also scale out in response to declining performance.

Remember that, in the hotel reservation system example, you increase the number of instances of the web app when you anticipate extra traffic because of a special event, a special offer, or because of seasonal fluctuations. You scale the system back when the demand drops.

In this exercise, you'll create an App Service plan and deploy a web app using this plan. You'll monitor the performance of the web app under load. You'll then scale out the app and verify that its performance has improved as a result.

The exercise uses a sample web app that implements a web API. The web API exposes HTTP POST and GET operations that create and retrieve customer bookings for a hotel reservations web site. The bookings aren't actually saved, and the GET operation simply retrieves dummy data.

The exercise also runs a client app that simulates a number of users issuing POST and GET operations simultaneously. This app provides the workload that's used to test the performance of the web app before and after scaling.

## Create an App Service plan and web app

### Important

You need your own Azure subscription to run this exercise and you may incur charges. If you don't already have an Azure subscription, create a [free account](#) before you begin

1. Sign in to the [Azure portal](#).
2. On the Azure portal menu or from the **Home** page, select **Create a resource**
3. Select **Web > Web App**
4. On the **Web App** page, enter the values in the following table.

### Note

The web app must have a unique name. We suggest using something like **<your name or initials>hotelsystem** Use this name wherever you see <your-webapp-name> in this exercise.

Property	Value
App name	<your-webapp-name >
Subscription	Select the Azure subscription you'd like to use for this exercise
Resource Group	Create a new resource group called <b>mslearn-scale</b>
OS	Windows
Publish	Code
App Service plan/Location	<i>Leave default</i>

5. Select **Create** and wait for the web app to be created.

## Build and deploy the web app

1. Open the Cloud Shell in the Azure portal. Run this command to download the source code for the hotel reservation system:

```
bash

git clone https://github.com/MicrosoftDocs/mslearn-hotel-reservation-system.git
```

Copy

2. Go to the `mslearn-hotel-reservation-system/src` folder:

```
bash

cd mslearn-hotel-reservation-system/src
```

3. Build the apps for the hotel system. There are two apps: a web app that implements the web API for the system and a client app that you'll use to load test the web app.

```
bash

dotnet build
```

4. Prepare the `HotelReservationSystem` web app for publishing:

```
bash

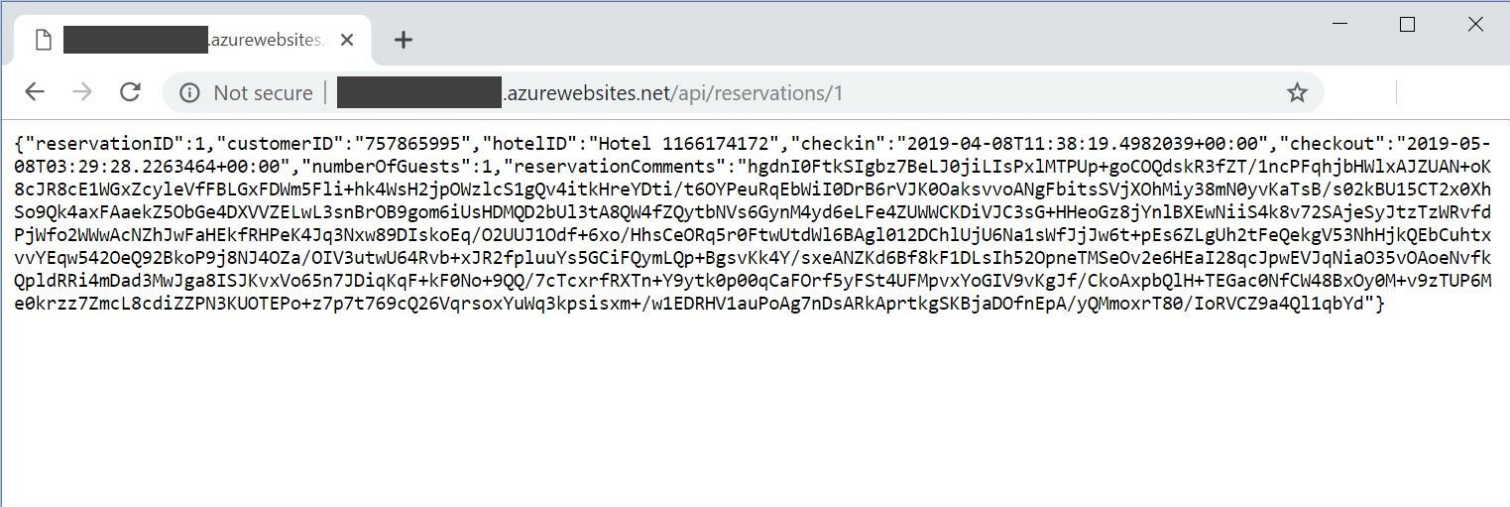
cd HotelReservationSystem
dotnet publish -o website
```

5. Go to the `website` folder, which contains the published files. Zip the files and deploy them to the web app that you created in the previous task. Replace `<your-webapp-name>` with the name of your web app.

```
bash

cd website
zip website.zip *
az webapp deployment source config-zip --src website.zip --name <your-webapp-name> --resource-group mslearn-scale
```

6. Use your web browser to go to `http://<your-webapp-name>.azurewebsites.net/api/reservations/1`. You should see a JSON document that contains the details for reservation number 1:



## Monitor the performance of the web app before scaling out

1. Return to the Cloud Shell and go to the `~/mslearn-hotel-reservation-system/src/HotelReservationSystemTestClient` folder:

```
bash

cd ~/mslearn-hotel-reservation-system/src/HotelReservationSystemTestClient
```

2. Edit the `App.config` file in this folder by using the code editor:

```
bash

code App.config
```

3. Uncomment the line that specifies the `ReservationsServiceURI` and replace the value with the URL of your web app. The file should like this example:

text	Copy
<pre>&lt;?xml version="1.0" encoding="utf-8" ? &gt; &lt;configuration &gt;   &lt;appSettings&gt;     &lt;add key="NumClients" value="100" /&gt;     &lt;add key="ReservationsServiceURI" value="https://&lt;your-webapp-name&gt;.azurewebsites.net/" /&gt;     &lt;add key="ReservationsServiceCollection" value="api/reservations" /&gt;   &lt;/appSettings&gt; &lt;/configuration &gt;</pre>	



Note

The `NumClients` setting in this file specifies the number of clients that will simultaneously try to connect to the web app and perform work. The work consists of creating a reservation and then running a query to fetch the details of the reservation. All the data used is fake. It's not actually persisted anywhere. Leave this value set to `100`.

4. Save the file and close the code editor.

5. Rebuild the test client app with the new configuration:

bash	Copy
<pre>dotnet build</pre>	

6. Run the client app. You'll see a number of messages appear as the clients start running, make reservations, and run queries. Allow the system to run for a couple of minutes. The responses will be slow, and soon the client requests will start to fail with HTTP 408 (Timeout) errors.

bash	Copy
<pre>dotnet run</pre>	



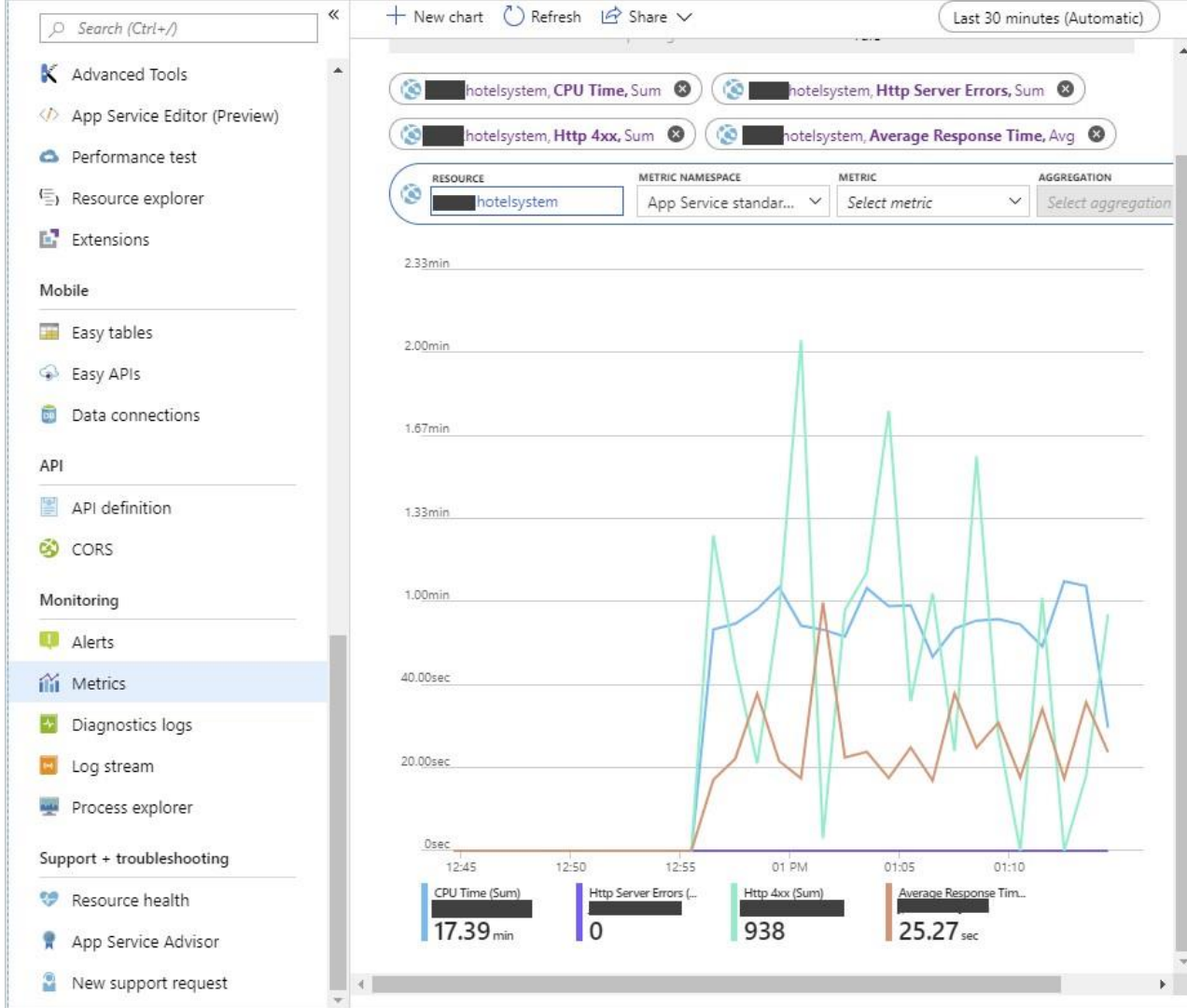
```
Bash
pv274e4QIf3Qa0btYyDyhT5mik1G1IqpygmZNT/UAwIHnJ+hwey9RyWEnw80QMVEsNF-iMI5CV495MavXQVBN8=
Client Client45 making reservation 1109836230
Client Client53 querying reservation: Reservation 351998021, CustomerID 795210174, HotelID Hotel 182
Comments: j7EEtVWwGOLUfhvEG022xSuPL42v1aCxFKqNbmBJMGMO0LXpCJ0upi3LMFbf+c7iKTHYZJBVOVgqOMkmUoGnnEdrKFf
4sTb8Z4CjR7s73oX/Bc6Qj1vY2H29DkiE7eEyoZkNeXSSGodZ/gZJNveuBC0G91fvUKqyC2ywwXGNZFD15Y0VApjd4RKk+3S6N/ol
eXWtVnDKSfxxzrq/s/eP3NivpzK806cAA3A+2tNBuE0k7QvWqf81tU+SIImNZ7IBY+E2er1GloAtqBLY3N8MdgxDys015AFTaaJuqA
ghHB7yONiFDj+KUMqWjtlp13cZuFox3ysaRUP9uqBk9hvtD/iMwY6ugz0Kh09MoP3Qqe4Cwfj91H6y4Ha0inGnVp4hk/45iYKy6V
offsWPL5noI8qjozoxnnah13mo4PEgBmz+95gZATqxDvyyUxEGzk07/hTfqjQg6PeHVL5h41GYIfCdGIaAfyrhFmJQuKZeQzIqCz
Client Client53 making reservation 1685198217
Client Client33 querying reservation: Reservation 589443144, CustomerID 1034084455, HotelID Hotel 21
Comments: LtR9w1xGqiyzdmOROY3qh80EJSD+qVkud7P57++DtDoUzwnNwRskAJYt78joFsvCzH2JuiEdYJNewAdx8+T0zS5dv9L
6VLHocD1Po1HEBMOi5Xng/jellYON7BPYMBM2teXa2Feq0t3mC4Rw4Hx/x6hu2cw90THSEGGcvQZJA/sfzvYYBayjOgmJuQMbrn2
UTiaf+vBQ8jzbTeLYoeSHf6VY07U+PXu9u3Wt7yd0YkdCbJN/np1LEfwBEKiGnDakyXv9+QowUgiwC+6mRIR9qy5bDFjS1JYC6Tt
Client Client33 making reservation 2102474681
Client Client43 querying reservation: Reservation 160354436, CustomerID 766230361, HotelID Hotel 345
Comments: oWps80CRnI4hGcYT6KyF3jdSSG4Y5Y516ebMqfwZonueMoazeGhzB+eFwy/qObR1bdqdz4UiyhNM03imtDptkITZVf
Drq+x4/BlgQDmiQhLgeccmD5KAP7+0EdDBKFyu1Mncregj22qHakPJiyqUk1lnAvtxvhkuFq79wNFPQOWN07hJigVMHwWcm3mu
WYCue34SOJGCEsb3obuUtDabXezRVPDiZuprW9fk+SV+AjmEj4xkPjZKTOjGYslw9otdk7Z+0dUq+9K/JuPhkN/hv54p10mtf40s
Client Client43 making reservation 1480437344
Client Client16 querying reservation: Reservation 272106587, CustomerID 1539781140, HotelID Hotel 13
Comments: xNsgr8mm/sdluXI3LzjfibvW9Qpmjr8fYdjeoPKOzD+O4TMKRwailioS4/yBREJkwqmucoAtXoRGA/HtaYTrcx9xwcl
p2cx6eAaya0TwbttNsS1Bh0afPHtjwWlByPseQvCvwCDFAwHtQqHeXCIxEXU/n3Io0Fy5ZXmb001QGKYKKn3LbE+tf+f++EQ20p0
q9ks8M/fq8F+biBVpQtKdNkVtMrja106+AA+p3WYK4A0mZWQYPhLv4eIbA6Rif3Bng3R6LDW7bh2nLyhxM5dWls8R7sn9et9tuB6d
SRF7/woT3yltq8P3GCq1vmm0jBdzLGB+XfH0Fr+LPAZL1QuI0TB1QzD9mK7L1L5Uelwh5cin3mMEsNiss+fuT0kKjQdY5XqRyzvg
1lVsVj7h7ecUkyYrZNHtNjVq67gyv6emsCRGICRuQwyJWPQ1rjMo0N3wMCyCnD6VRK2hQCWabFSG+9BxiET5w9HRj5XYTXPF1RhyL
Client Client16 making reservation 1072786692
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Client Client8 querying reservation: Reservation 798222214, CustomerID 609864951, HotelID Hotel 8613
Comments: DK6pMVElQvM81tVef2zxN2v3Gy1Jve3jotCDZkoKmxgYDATOz+R0h63Us0CPghD40Pg9sCgmDGFuEhkVERBc8zSjTY
6900svN0kjyGzRC8wziUVsPB0JJiD0QpMnFIWag8MwTZY+DUakUoyPaP5i0uHi2XRqK8BZ0k3TBImLmuVPEVWVHORxkC+Lkm2Qc
7ii6KpVfRaf
Client Client8 making reservation 993123983
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
Error making reservation: Response status code does not indicate success: 408 (Request Timeout).
```

7. In the Azure portal, go to the pane for your web app (not the service plan). Under **Monitoring**, select **Metrics**.

8. Add the following metrics to the chart, set the time range to **Last 30 minutes** and then pin the chart to the current dashboard.

- CPU Time. Select the Sum aggregation.
- Http Server Errors. Select the Sum aggregation.
- Http 4xx. Select the Sum aggregation.
- Average Response Time. Select the Avg aggregation.

9. Allow the system to run for five minutes to stabilize, and then note the CPU Time, the number of HTTP 4xx errors, and the average response time. You should see a significant number of HTTP 4xx errors (these are HTTP 408 Timeout errors), and that the average response time is several seconds. You might see the occasional HTTP server error, depending on how the web server is coping with the burden.



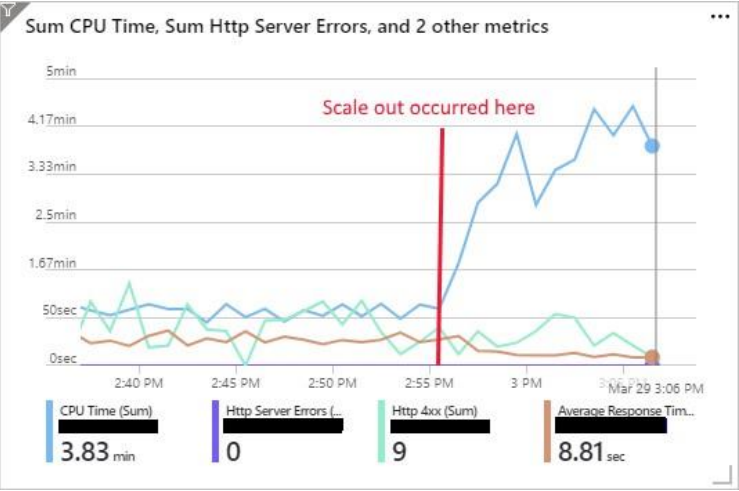
10. Leave the client app running while you perform the next task.

## Scale out the web app and verify the performance improvement

1. In the Azure portal, in the pane for your web app, under **Settings**, select **Scale out (App Service Plan)**
2. On the **Configure** page, set the **Instance count** to **5**, and then select **Save**.

3. Switch to the Cloud Shell that's running the client app. You should see fewer requests failing with errors, though you'll still see some that time out.
4. Run the app for another five minutes. Then go to the chart that shows the metrics for the app on the dashboard in the Azure portal. You should see that the CPU time has increased dramatically because there's now five times more CPU power available. The average response time should have dropped, and

the number of HTTP 4xx errors should also have decreased. The following chart shows a typical set of results. The point at which scale out occurred is noted.



- 5. If you want to experiment some more, try increasing the instance count for the App Service plan to 10. Ten is the maximum number of instances supported by the S1 tier. You should notice a further increase in CPU time, and a corresponding drop in response time and HTTP 4xx errors.
- 6. Return to the Cloud Shell that's running the client app. Select Enter to stop the app.
- 7. In the Azure portal, set the instance count for the App Service plan back to 1.

Next unit: Scale up a web app

Continue



# Scale up a web app

5 minutes

Scaling out enables you to run more instances of a web app, but the resources available to each instance are determined by the pricing tier used by the App Service plan that hosts the web service. Each pricing tier specifies the computing power provided, together with the memory and maximum number of instances that can be created.

If you initially deploy a web app using a relatively cheap pricing tier, you might find the resources are sufficient to start with. But the resources might become too limited if demand for your web service grows, or if you add features that require more power. In this case, you can scale up to a more powerful pricing tier.

In the hotel reservation system, you've noticed a steady increase in the number of visitors, beyond the variations caused by special offers or events. And your company is adding more features to the web app that require additional resources. You're nearing the scale-out limits of your current App Service plan pricing tier, so you need to scale up to a tier that provides more instances and more powerful hardware.

In this unit, you'll learn how to scale up the web app to meet the increasing resource requirements.

## App Service plan pricing tiers and hardware levels

The different pricing tiers available for App Service plans offer various levels or resources. The Basic, Standard, and Premium tiers are based on *Dv2-Series* VMs that have different amounts of memory and IO capacity. The PremiumV2 and Isolated tiers are based on *Dv2-Series* VMs. Each of these tiers has three hardware levels, roughly corresponding to 1, 2, and 4 CPUs. For detailed information about the pricing tiers and hardware levels, see [App Service pricing](#).

## Scale up a web app

You scale an App Service plan up and down by changing the pricing tier and hardware level that it runs on. You can start with the Free tier and scale up as needed according to your requirements. This process is manual. You can also scale down again if you no longer need the resources associated with a particular tier.

Scaling up can cause an interruption in service to client apps running at the time. They might need to disconnect from the service and reconnect if the scale-up occurs during an active call to the web app. And new connections might be rejected until scaling finishes. Also, scaling up can cause the outgoing IP addresses for the web app to change. If your web app depends on other services that have firewalls restricting incoming traffic, you'll need to reconfigure these services.

As with scale-out, you should monitor the performance of your system to ensure that scaling up (or down) has the desired effect. It's also important to understand that scale up and scale out can work cooperatively together. If you have scaled out to the maximum number of instances available for your pricing tier, you must scale up before you can scale out further.

Next unit: Exercise - Scale up a web app

Continue

# Exercise - Scale up a web app

15 minutes

Scaling up provides more powerful resources for running a web app. It also increases the number of instances available for scaling out.

In the hotel reservation system, you should scale out to handle the increasing number of visitors to the web app. Scaling up will enable you to scale out further. Scaling up will also likely be necessary to support the new functionality that will be added to the web app.

In this exercise, you'll scale up the hotel reservation system web app that you deployed earlier. You'll run the same test client application that you used before, and monitor the performance of the web app.

## Examine the current pricing tier for the web app

### Important

You need your own Azure subscription to run this exercise and you may incur charges. If you don't already have an Azure subscription, create a [free account](#) before you begin

1. Sign in to the [Azure portal](#).
2. On the Azure portal menu or from the **Home** page, select **All resources** and then go to your App Service plan.
3. Under **Settings**, select **Scale up (App Service plan)** You should see details of the pricing tier for your App Service plan. The pricing tier is S1, which provides 100 Azure Compute Units and 1.75 GB of memory running on an A-Series virtual machine.

webappplan - Scale up (App Service plan)

App Service plan

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

Apps

File system storage

Networking

Scale up (App Service plan)

Scale out (App Service plan)

Resource explorer

Properties

Locks

Export template

Monitoring

Alerts

Metrics

Support + troubleshooting

Resource health

New support request

Dev / Test

For less demanding workloads

Production

For most production workloads

Isolated

Advanced networking and scale

Recommended pricing tiers

S1

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
55.43 GBP/Month (Estimated)

P1V2

210 total ACU  
3.5 GB memory  
Dv2-Series compute equivalent  
110.86 GBP/Month (Estimated)

P2V2

420 total ACU  
7 GB memory  
Dv2-Series compute equivalent  
221.79 GBP/Month (Estimated)

P3V2

840 total ACU  
14 GB memory  
Dv2-Series compute equivalent  
443.57 GBP/Month (Estimated)

See additional options

Included features

Every app hosted on this App Service plan will have access to these features:

Custom domains / SSL

Configure and purchase custom domains with SNI and IP SSL bindings

Auto scale

Up to 10 instances. Subject to availability.

Staging slots

Up to 5 staging slots to use for testing and deployments before swapping them into production.

Daily backups

Backup your app 10 times daily.

Traffic manager

Apply

Included hardware

Every instance of your App Service plan will include the following hardware configuration:

Azure Compute Units (ACU)

Dedicated compute resources used to run applications deployed in the App Service Plan. [Learn more](#)

Memory

Memory per instance available to run applications deployed and running in the App Service plan.

Storage

50 GB disk storage shared by all apps deployed in the App Service plan.

# Run the test client app

1. In the Cloud Shell window on the right side of the screen, go to the `/mslearn-hotel-reservation-system/src/HotelReservationSystemTestClient` folder:

```
bash
cd ~/mslearn-hotel-reservation-system/src/HotelReservationSystemTestClient
```

2. Run the client app. Allow the system to run for a couple minutes. As at the start of the previous exercise, the responses will be slow, and the client requests will soon start to fail with HTTP 408 (Timeout) errors.

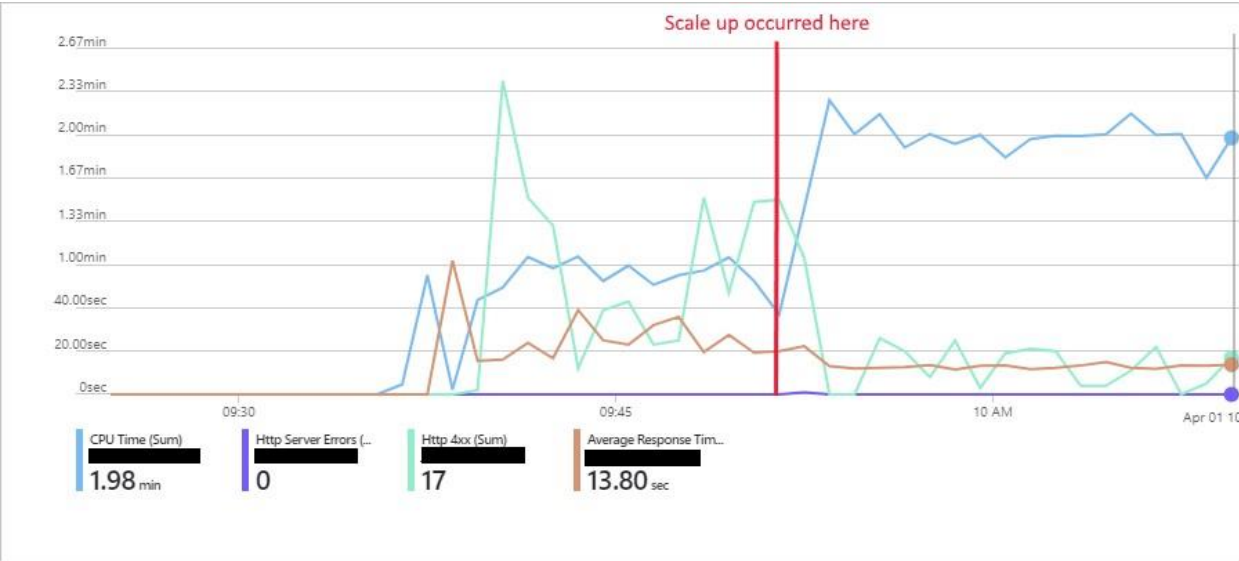
```
bash
dotnet run
```

3. Keep running the app. Wait for another five minutes. Then go to the chart that shows the metrics for the web app on the dashboard in the Azure portal. As in the previous exercise, you should see that the statistics indicate a relatively slow response time and many HTTP 4xx errors.

## Scale up the web app and monitor the results

1. In the Azure portal, return to the page for the App Service plan.
2. Under **Settings**, select **Scale up (App Service plan)**
3. Select the **P2V2** pricing tier and then select **Apply**. This pricing tier gives you 420 ACU (more than four times the power of the S1 pricing tier), and 7 GB of memory, running on a Dv2-Series VM. But this VM costs four times the cost of running the S1 pricing tier.
4. Wait for another five minutes, and then view the performance chart on the dashboard in the Azure portal.
5. At the time of the system scale-up, you might notice some additional HTTP server errors. These errors are caused by ongoing client requests that were aborted when the system switched hardware. After the scale-up, the CPU time jumps because more processors are available. You might not notice the same drop in response time that you saw when scaling out. This is because you're still using only a single instance, so requests aren't being load balanced as they were when you scaled out. But you now have the opportunity to scale out across more instances (20) than you had before.

The chart in this image shows an example of the performance metrics for the web app. The point at which the system was scaled up is noted.



6. Return to the Cloud Shell that's running the client app. Select Enter to stop the app.

### Next unit: Summary