100 XP

Introduction

3 minutes

Autoscaling enables a system to adjust the resources required to meet the varying demand from users, while controlling the costs associated with these resources. You can use autoscaling with many Azure services, including web applications. Autoscaling requires you to configure autoscale rules that specify the conditions under which resources should be added or removed.

Learning objectives

After completing this module, you'll be able to:

- Identify scenarios for which autoscaling is an appropriate solution
- Create autoscaling rules for a web app
- Monitor the effects of autoscaling

Next unit: Examine autoscale factors

Continue

100 XP

Examine autoscale factors

3 minutes

Autoscaling can be triggered according to a schedule, or by assessing whether the system is running short on resources. For example, autoscaling could be triggered if CPU utilization grows, memory occupancy increases, the number of incoming requests to a service appears to be surging, or some combination of factors.

What is autoscaling?

Autoscaling is a cloud system or process that adjusts available resources based on the current demand. Autoscaling performs scaling *in and out*, as opposed to scaling *up and down*.

Azure App Service Autoscaling

Autoscaling in Azure App Service monitors the resource metrics of a web app as it runs. It detects situations where additional resources are required to handle an increasing workload, and ensures those resources are available before the system becomes overloaded.

Autoscaling responds to changes in the environment by adding or removing web servers and balancing the load between them. Autoscaling doesn't have any effect on the CPU power, memory, or storage capacity of the web servers powering the app, it only changes the number of web servers.

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.

Define your autoscaling rules carefully. For example, a Denial of Service attack will likely result in a large-scale influx of incoming traffic. Trying to handle a surge in requests caused by a DoS attack would be fruitless and expensive. These requests aren't genuine, and should be discarded rather

than processed. A better solution is to implement detection and filtering of requests that occur during such an attack before they reach your service.

When should you consider autoscaling?

Autoscaling provides elasticity for your services. It's a suitable solution when hosting any application when you can't easily predict the workload in advance, or when the workload is likely to vary by date or time. For example, you might expect increased/reduced activity for a business app during holidays.

Autoscaling improves availability and fault tolerance. It can help ensure that client requests to a service won't be denied because an instance is either not able to acknowledge the request in a timely manner, or because an overloaded instance has crashed.

Autoscaling works by adding or removing web servers. If your web apps perform resource-intensive processing as part of each request, then autoscaling might not be an effective approach. In these situations, manually scaling up may be necessary. For example, if a request sent to a web app involves performing complex processing over a large dataset, depending on the instance size, this single request could exhaust the processing and memory capacity of the instance.

Autoscaling isn't the best approach to handling long-term growth. You might have a web app that starts with a small number of users, but increases in popularity over time. 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.

The number of instances of a service is also a factor. You might expect to run only a few instances of a service most of the time. However, in this situation, your service will always be susceptible to downtime or lack of availability whether autoscaling is enabled or not. The fewer the number of instances initially, the less capacity you have to handle an increasing workload while autoscaling spins up additional instances.

Next unit: Identify autoscale factors

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Identify autoscale factors

3 minutes

Autoscaling enables you to specify the conditions under which a web app should be scaled out, and back in again. Effective autoscaling ensures sufficient resources are available to handle large volumes of requests at peak times, while managing costs when the demand drops.

You can configure autoscaling to detect when to scale in and out according to a combination of factors, based on resource usage. You can also configure autoscaling to occur according to a schedule.

In this unit, you'll learn how to specify the factors that can be used to autoscale a service.

Autoscaling and the App Service Plan

Autoscaling is a feature of the App Service Plan used by the web app. When the web app scales out, Azure starts new instances of the hardware defined by the App Service Plan to the app.

To prevent runaway autoscaling, an App Service Plan has an instance limit. Plans in more expensive pricing tiers have a higher limit. Autoscaling cannot create more instances than this limit.

> **Note**
>
> Not all App Service Plan pricing tiers support autoscaling.

Autoscale conditions

You indicate how to autoscale by creating autoscale conditions. Azure provides two options for autoscaling:

- Scale based on a metric, such as the length of the disk queue, or the number of HTTP requests awaiting processing.

- Scale to a specific instance count according to a schedule. For example, you can arrange to scale out at a particular time of day, or on a specific date or day of the week. You also specify an end date, and the system will scale back in at this time.

Scaling to a specific instance count only enables you to scale out to a defined number of instances. If you need to scale out incrementally, you can combine metric and schedule-based autoscaling in the same autoscale condition. So, you could arrange for the system to scale out if the number of HTTP requests exceeds some threshold, but only between certain hours of the day.

You can create multiple autoscale conditions to handle different schedules and metrics. Azure will autoscale your service when any of these conditions apply. An App Service Plan also has a default condition that will be used if none of the other conditions are applicable. This condition is always active and doesn't have a schedule.

Metrics for autoscale rules

Autoscaling by metric requires that you define one or more autoscale rules. An autoscale rule specifies a metric to monitor, and how autoscaling should respond when this metric crosses a defined threshold. The metrics you can monitor for a web app are:

- **CPU Percentage**. This metric is an indication of the CPU utilization across all instances. A high value shows that instances are becoming CPU-bound, which could cause delays in processing client requests.
- **Memory Percentage**. This metric captures the memory occupancy of the application across all instances. A high value indicates that free memory could be running low, and could cause one or more instances to fail.
- **Disk Queue Length**. This metric is a measure of the number of outstanding I/O requests across all instances. A high value means that disk contention could be occurring.
- **Http Queue Length**. This metric shows how many client requests are waiting for processing by the web app. If this number is large, client requests might fail with HTTP 408 (Timeout) errors.
- **Data In**. This metric is the number of bytes received across all instances.
- **Data Out**. This metric is the number of bytes sent by all instances.

You can also scale based on metrics for other Azure services. For example, if the web app processes requests received from a Service Bus Queue, you might want to spin up additional instances of a web app if the number of items held in an Azure Service Bus Queue exceeds a critical length.

How an autoscale rule analyzes metrics

Autoscaling works by analyzing trends in metric values over time across all instances. Analysis is a multi-step process.

In the first step, an autoscale rule aggregates the values retrieved for a metric for all instances across a period of time known as the *time grain*. Each metric has its own intrinsic time grain, but in most cases this period is 1 minute. The aggregated value is known as the *time aggregation*. The options available are *Average*, *Minimum*, *Maximum*, *Total*, *Last*, and *Count*.

An interval of one minute is a very short interval in which to determine whether any change in metric is long-lasting enough to make autoscaling worthwhile. So, an autoscale rule performs a second step that performs a further aggregation of the value calculated by the *time aggregation* over a longer, user-specified period, known as the *Duration*. The minimum *Duration* is 5 minutes. If the *Duration* is set to 10 minutes for example, the autoscale rule will aggregate the 10 values calculated for the *time grain*.

The aggregation calculation for the *Duration* can be different from that of the *time grain*. For example, if the *time aggregation* is *Average* and the statistic gathered is *CPU Percentage* across a one-minute *time grain*, each minute the average CPU percentage utilization across all instances for that minute will be calculated. If the *time grain statistic* is set to *Maximum*, and the *Duration* of the rule is set to 10 minutes, the maximum of the 10 average values for the CPU percentage utilization will be used to determine whether the rule threshold has been crossed.

Autoscale actions

When an autoscale rule detects that a metric has crossed a threshold, it can perform an autoscale action. An autoscale action can be *scale-out* or *scale-in*. A scale-out action increases the number of instances, and a scale-in action reduces the instance count. An autoscale action uses an operator (such as *less than*, *greater than*, *equal to*, and so on) to determine how to react to the threshold. Scale-out actions typically use the *greater than* operator to compare the metric value to the threshold. Scale-in actions tend to compare the metric value to the threshold with the *less than* operator. An autoscale action can also set the instance count to a specific level, rather than incrementing or decrementing the number available.

An autoscale action has a *cool down* period, specified in minutes. During this interval, the scale rule won't be triggered again. This is to allow the system to stabilize between autoscale events. Remember that it takes time to start up or shut down instances, and so any metrics gathered

might not show any significant changes for several minutes. The minimum cool down period is five minutes.

Pairing autoscale rules

You should plan for scaling-in when a workload decreases. Consider defining autoscale rules in pairs in the same autoscale condition. One autoscale rule should indicate how to scale the system out when a metric exceeds an upper threshold. Then other rule should define how to scale the system back in again when the same metric drops below a lower threshold.

Combining autoscale rules

A single autoscale condition can contain several autoscale rules (for example, a scale-out rule and the corresponding scale-in rule). However, the autoscale rules in an autoscale condition don't have to be directly related. You could define the following four rules in the same autoscale condition:

- If the HTTP queue length exceeds 10, scale out by 1
- If the CPU utilization exceeds 70%, scale out by 1
- If the HTTP queue length is zero, scale in by 1
- If the CPU utilization drops below 50%, scale in by 1

When determining whether to scale out, the autoscale action will be performed if **any** of the scale-out rules are met (HTTP queue length exceeds 10 **or** CPU utilization exceeds 70%). When scaling in, the autoscale action will run **only if all** of the scale-in rules are met (HTTP queue length drops to zero **and** CPU utilization falls below 50%). If you need to scale in if only one the scale-in rules are met, you must define the rules in separate autoscale conditions.

Next unit: Enable autoscale in App Service

Continue >

---

100 XP

Enable autoscale in App Service

3 minutes

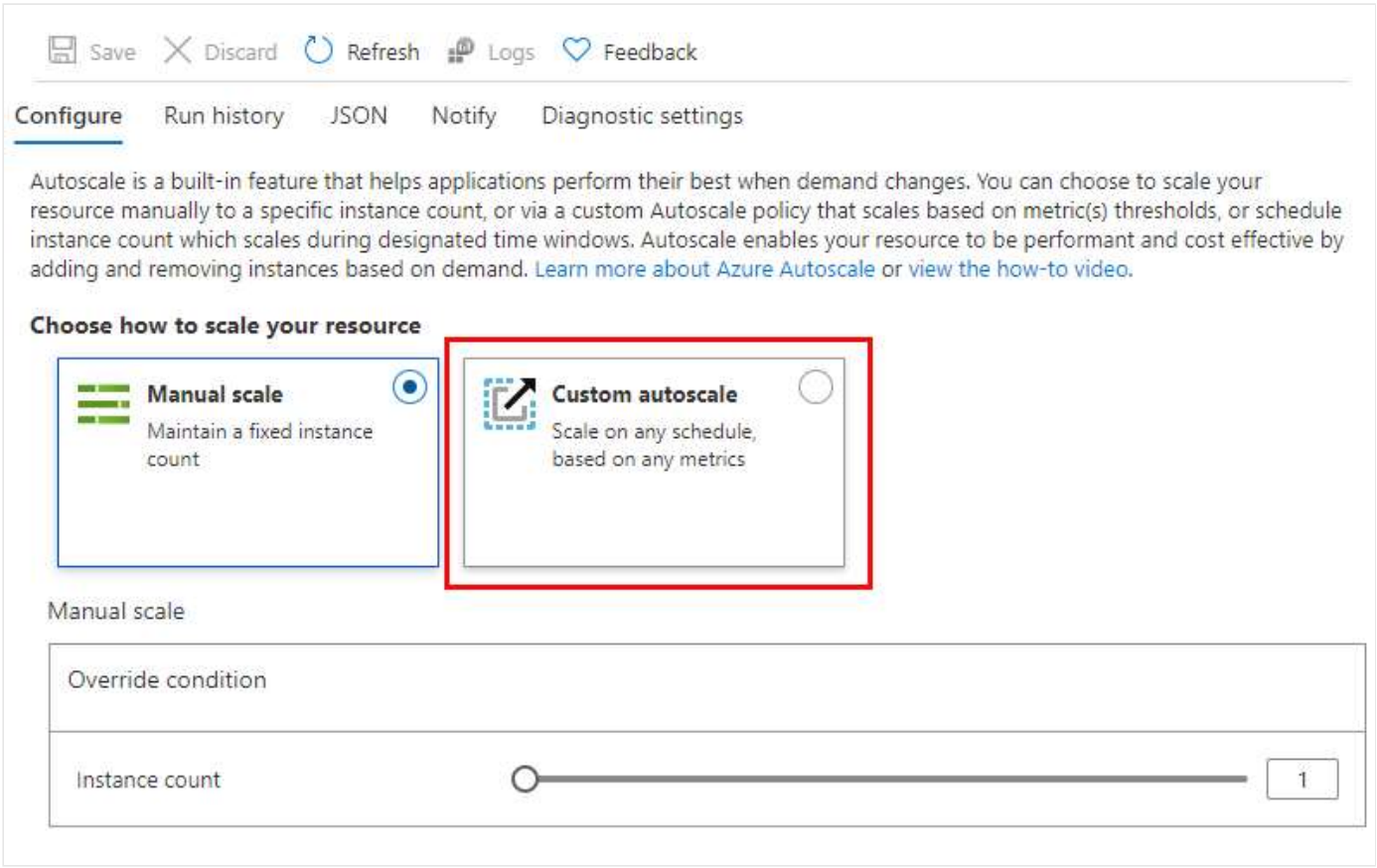
In this unit, you will learn how to enable autoscaling, create autoscale rules, and monitor autoscaling activity

Enable autoscaling

To get started with autoscaling navigate to your App Service plan in the Azure portal and select **Scale out (App Service plan)** in the **Settings** group in the left navigation pane.

> **Note**
>
> Not all pricing tiers support autoscaling. The development pricing tiers are either limited to a single instance (the **F1** and **D1** tiers), or they only provide manual scaling (the **B1** tier). If you've selected one of these tiers, you must first scale up to the **S1** or any of the **P** level production tiers.

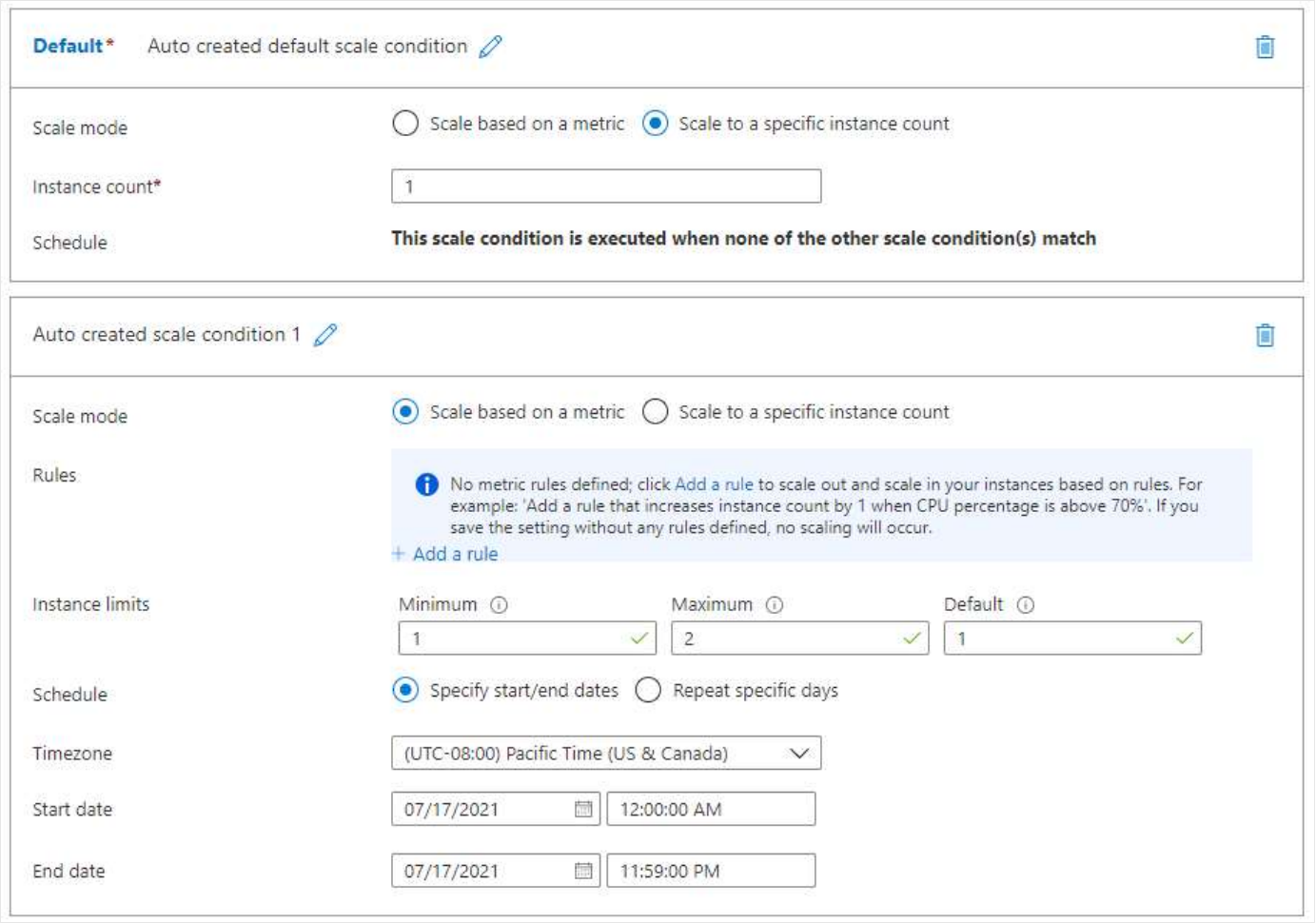
By default, an App Service Plan only implements manual scaling. Selecting **Custom autoscale** reveals condition groups you can use to manage your scale settings.



Add scale conditions

Once you enable autoscaling, you can edit the automatically created default scale condition, and you can add your own custom scale conditions. Remember that each scale condition can either scale based on a metric, or scale to a specific instance count.

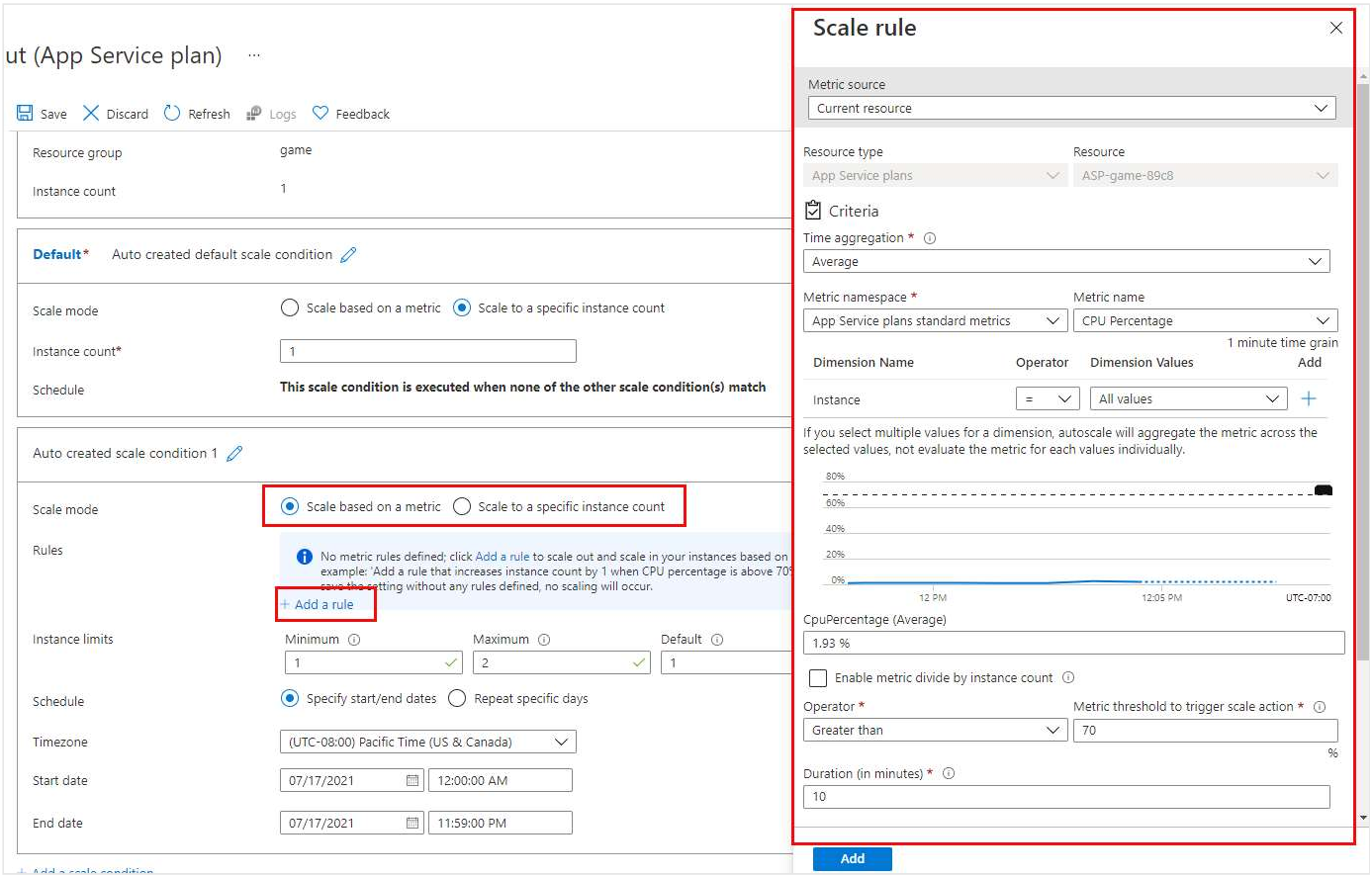
The Default scale condition is executed when none of the other scale conditions are active.



A metric-based scale condition can also specify the minimum and maximum number of instances to create. The maximum number can't exceed the limits defined by the pricing tier. Additionally, all scale conditions other than the default may include a schedule indicating when the condition should be applied.

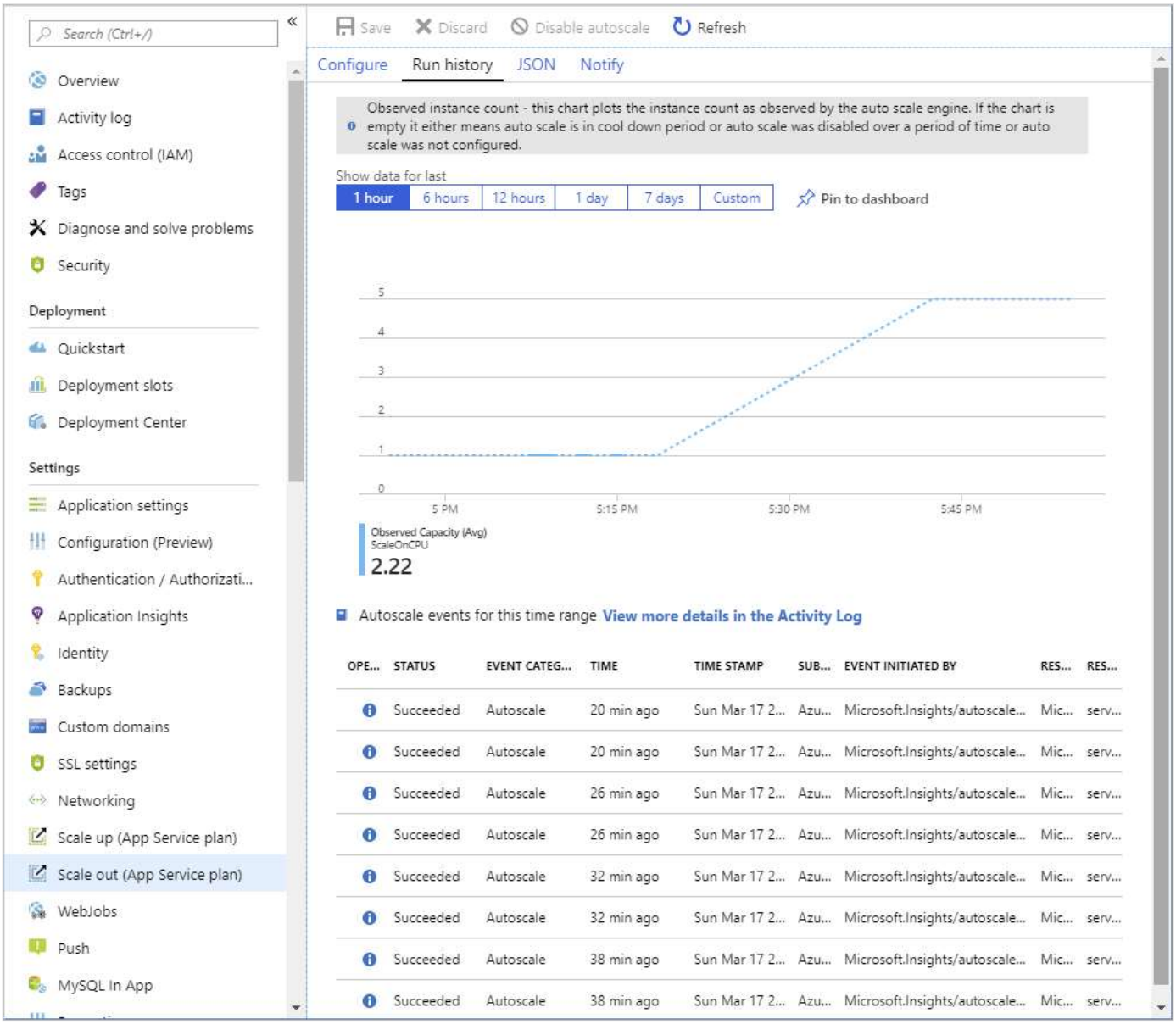
Create scale rules

A metric-based scale condition contains one or more scale rules. You use the **Add a rule** link to add your own custom rules. You define the criteria that indicate when a rule should trigger an autoscale action, and the autoscale action to be performed (scale out or scale in) using the metrics, aggregations, operators, and thresholds described earlier.

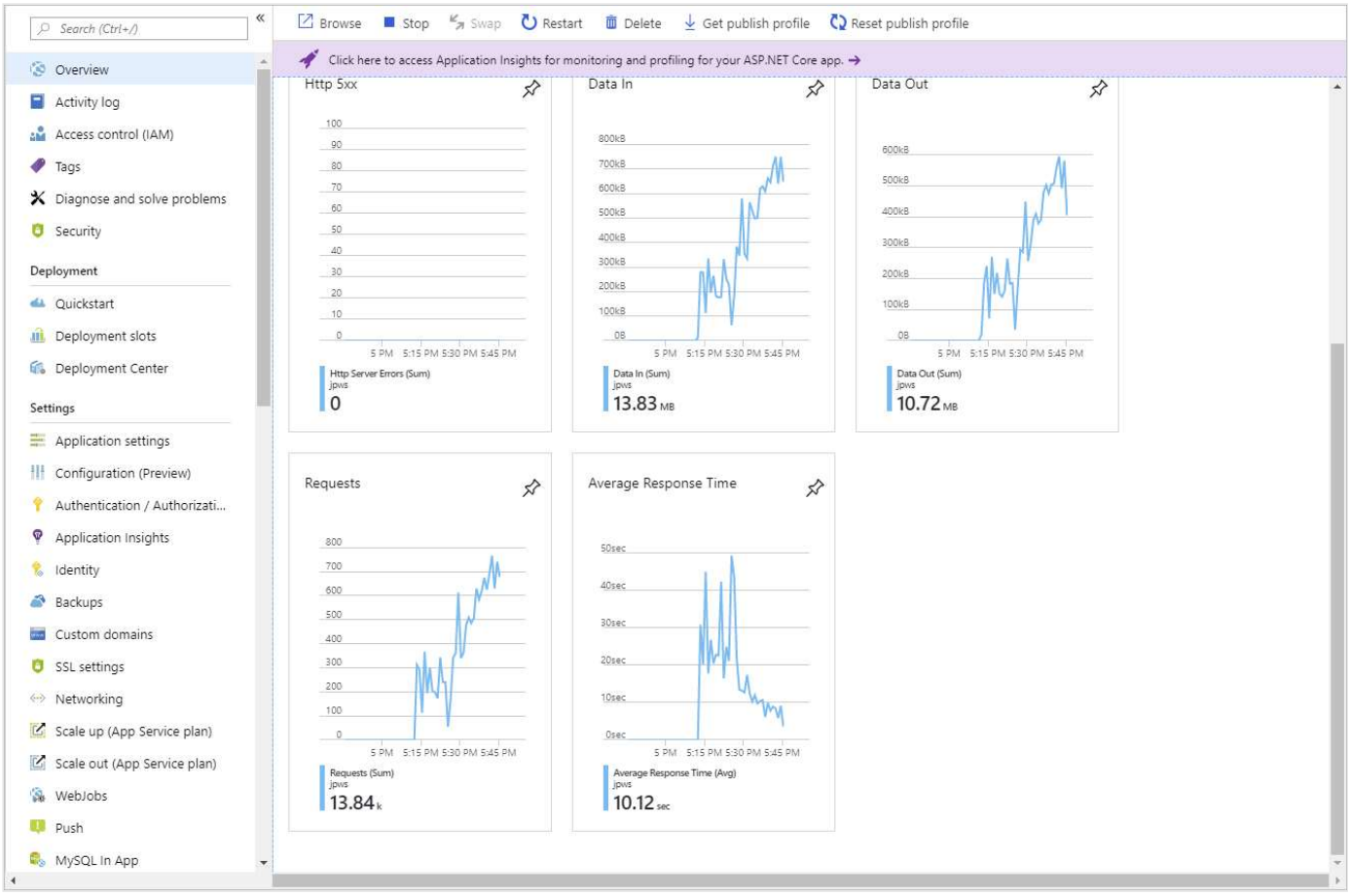


Monitor autoscaling activity

The Azure portal enables you to track when autoscaling has occurred through the **Run history** chart. This chart shows how the number of instances varies over time, and which autoscale conditions caused each change.



You can use the **Run history** chart in conjunction with the metrics shown on the **Overview** page to correlate the autoscaling events with resource utilization.



Next unit: Explore autoscale best practices

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore autoscale best practices

3 minutes

If you're not following good practices when creating autoscale settings you can create conditions that lead to undesirable results. In this unit you will learn how to avoid creating rules that conflict with each other.

Autoscale concepts

- An autoscale setting scales instances horizontally, which is *out* by increasing the instances and *in* by decreasing the number of instances. An autoscale setting has a maximum, minimum, and default value of instances.
- An autoscale job always reads the associated metric to scale by, checking if it has crossed the configured threshold for scale-out or scale-in.
- All thresholds are calculated at an instance level. For example, "scale out by one instance when average CPU > 80% when instance count is 2", means scale-out when the average CPU across all instances is greater than 80%.
- All autoscale successes and failures are logged to the Activity Log. You can then configure an activity log alert so that you can be notified via email, SMS, or webhooks whenever there is activity.

Autoscale best practices

Use the following best practices as you create your autoscale rules.

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

If you have a setting that has minimum=2, maximum=2 and the current instance count is 2, no scale action can occur. Keep an adequate margin between the maximum and minimum instance

counts, which are inclusive. Autoscale always scales between these limits.

Choose the appropriate statistic for your diagnostics metric

For diagnostics metrics, you can choose among *Average*, *Minimum*, *Maximum* and *Total* as a metric to scale by. The most common statistic is Average.

Choose the thresholds carefully for all metric types

We recommend carefully choosing different thresholds for scale-out and scale-in based on practical situations.

We *do not recommend* autoscale settings like the examples below with the same or very similar threshold values for out and in conditions:

- Increase instances by 1 count when Thread Count >= 600
- Decrease instances by 1 count when Thread Count <= 600

Let's look at an example of what can lead to a behavior that may seem confusing. Consider the following sequence.

1. Assume there are two instances to begin with and then the average number of threads per instance grows to 625.
2. Autoscale scales out adding a third instance.
3. Next, assume that the average thread count across instance falls to 575.
4. Before scaling in, autoscale tries to estimate what the final state will be if it scaled in. For example, 575 x 3 (current instance count) = 1,725 / 2 (final number of instances when scaled in) = 862.5 threads. This means autoscale would have to immediately scale-out again even after it scaled in, if the average thread count remains the same or even falls only a small amount. However, if it scaled out again, the whole process would repeat, leading to an infinite loop.
5. To avoid this situation (termed "flapping"), autoscale does not scale in at all. Instead, it skips and reevaluates the condition again the next time the service's job executes. This can confuse many people because autoscale wouldn't appear to work when the average thread count was 575.

Estimation during a scale-in is intended to avoid "flapping" situations, where scale-in and scale-out actions continually go back and forth. Keep this behavior in mind when you choose the same thresholds for scale-out and in.

We recommend choosing an adequate margin between the scale-out and in thresholds. As an example, consider the following better rule combination.

- Increase instances by 1 count when CPU% >= 80
- Decrease instances by 1 count when CPU% <= 60

In this case

1. Assume there are 2 instances to start with.
2. If the average CPU% across instances goes to 80, autoscale scales out adding a third instance.
3. Now assume that over time the CPU% falls to 60.
4. Autoscale's scale-in rule estimates the final state if it were to scale-in. For example, 60 x 3 (current instance count) = 180 / 2 (final number of instances when scaled in) = 90. So autoscale does not scale-in because it would have to scale-out again immediately. Instead, it skips scaling in.
5. The next time autoscale checks, the CPU continues to fall to 50. It estimates again - 50 x 3 instance = 150 / 2 instances = 75, which is below the scale-out threshold of 80, so it scales in successfully to 2 instances.

Considerations for scaling when multiple rules are configured in a profile

There are cases where you may have to set multiple rules in a profile. The following set of autoscale rules are used by services when multiple rules are set.

On *scale-out*, autoscale runs if any rule is met. On *scale-in*, autoscale require all rules to be met.

To illustrate, assume that you have the following four autoscale rules:

- If CPU < 30 %, scale-in by 1
- If Memory < 50%, scale-in by 1
- If CPU > 75%, scale-out by 1
- If Memory > 75%, scale-out by 1

Then the follow occurs:

- If CPU is 76% and Memory is 50%, we scale-out.
- If CPU is 50% and Memory is 76% we scale-out.

On the other hand, if CPU is 25% and memory is 51% autoscale does not scale-in. In order to scale-in, CPU must be 29% and Memory 49%.

Always select a safe default instance count

The default instance count is important because autoscale scales your service to that count when metrics are not available. Therefore, select a default instance count that's safe for your workloads.

Configure autoscale notifications

Autoscale will post to the Activity Log if any of the following conditions occur:

- Autoscale issues a scale operation
- Autoscale service successfully completes a scale action
- Autoscale service fails to take a scale action.
- Metrics are not available for autoscale service to make a scale decision.
- Metrics are available (recovery) again to make a scale decision.

You can also use an Activity Log alert to monitor the health of the autoscale engine. In addition to using activity log alerts, you can also configure email or webhook notifications to get notified for successful scale actions via the notifications tab on the autoscale setting.

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

200 XP

Knowledge check

3 minutes

Check your knowledge

1. Which of these statements best describes autoscaling?

- ○ Autoscaling requires an administrator to actively monitor the workload on a system.
- ◉ Autoscaling is a scale out/scale in solution.

  ✔ **That's correct. The system can scale out when specified resource metrics indicate increasing usage, and scale in when these metrics drop.**
- ○ Scaling up/scale down provides better availability than autoscaling.

2. Which of these scenarios is a suitable candidate for autoscaling?

- ○ The number of users requiring access to an application varies according to a regular schedule. For example, more users use the system on a Friday than other days of the week.

  ✔ **That's correct. Changes in application load that are predictable are good candidates for autoscaling.**
- ○ The system is subject to a sudden influx of requests that grinds your system to a halt.
- ◉ Your organization is running a promotion and expects to see increased traffic to their web site for the next couple of weeks.

  ✘ **That's incorrect. Manual scaling is a better option here since this is a one-off event with a known duration.**

3. There are multiple rules in an autoscale profile. Which of the following scale operations will run if any of the rule conditions are met?

- (●) scale-out

  ✔ That's correct. Scale-out operations will trigger if any of the rule conditions are met.

- ( ) scale-in
- ( ) scale-out/in

Next unit: Summary

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Summary

3 minutes

In this module, you learned how to:

- Identify scenarios for which autoscaling is an appropriate solution
- Create autoscaling rules for a web app
- Monitor the effects of autoscaling

Module complete:

Continue to next module

How are we doing? ☆ ☆ ☆ ☆ ☆