```
2024-05-29T16:49:16 Welcome, you are now connected to log-streaming
service.Starting Log Tail -n 10 of existing logs ----
/appsvctmp/volatile/logs/runtime/container.log
2024-05-29T16:18:34.7804331Z As of "05/29/2024 16:18:31 +00:00", the
heartbeat has been running for "00:00:03.1849740" which is longer than
"00:00:01". This could be caused by thread pool starvation.
2024-05-29T16:41:15.4110451Z  [40m [1m [33mwarn [39m [22m [49m:
Microsoft.AspNetCore.Server.Kestrel[22]
2024-05-29T16:41:15.4238881Z As of "05/29/2024 16:41:12 +00:00", the
heartbeat has been running for "00:00:02.7675978" which is longer than
"00:00:01". This could be caused by thread pool starvation.
2024-05-29T16:42:21.8657532Z  [40m [1m [33mwarn [39m [22m [49m:
Microsoft.AspNetCore.Server.Kestrel[22]
2024-05-29T16:42:21.8681997Z As of "05/29/2024 16:42:19 +00:00", the
heartbeat has been running for "00:00:02.4450998" which is longer than
"00:00:01". This could be caused by thread pool starvation.
2024-05-29T16:42:54.1538186Z  [40m [1m [33mwarn [39m [22m [49m:
Microsoft.AspNetCore.Server.Kestrel[22]
2024-05-29T16:42:54.1558907Z As of "05/29/2024 16:42:51 +00:00", the
heartbeat has been running for "00:00:02.2874069" which is longer than
"00:00:01". This could be caused by thread pool starvation.
2024-05-29T16:47:16.9137683Z  [40m [1m [33mwarn [39m [22m [49m:
Microsoft.AspNetCore.Server.Kestrel[22]
2024-05-29T16:47:17.0796217Z As of "05/29/2024 16:47:14 +00:00", the
heartbeat has been running for "00:00:02.4238657" which is longer than
"00:00:01". This could be caused by thread pool starvation.
2024-05-29T16:48:51.2211266Z system test:::Ending Log Tail of existing logs -
--Starting Live Log Stream ---
2024-05-29T16:49:28.6098200Z system test:::
2024-05-29T16:49:29.7726535Z system test:::
2024-05-29T16:49:41.1488077Z system test:::
2024-05-29T16:49:44.7946698Z system test:::
```

Azure App Service troubleshooting documentation

Welcome to Azure App Service troubleshooting. These articles explain how to determine, diagnose, and fix issues that you might encounter when you use Azure Monitor. In the navigation pane on the left, browse through the article list or use the search box to find issues and solutions. Troubleshooting web apps c HOW-TO GUIDE FAQ about creating or deleting web apps Capture memory dumps on the Azure App Service platform Article • 06/22/2023 This article provides guidance about Microsoft Azure App Service debugging features for capturing memory dumps. The capture method that you use is dictated by the scenario in which you capture a memory dump for troubleshooting a performance or availability issue. For example, capturing a memory dump is different for a process that's experiencing excessive memory consumption than for a process that's throwing exceptions or responding slowly. The process in this context is the Internet Information Services (IIS) worker process (W3WP, which runs as w3wp.exe). The following table provides recommendations about the commands that each App Service feature runs to generate a memory dump. There are so many approaches to capturing a memory dump that the process might be confusing. If you're already proficient in capturing a W3WP memory dump, this information isn't intended to change your approach. Instead, we hope to provide guidance for inexperienced users

who haven't yet developed a preference. Scenario Azure App Service debugging feature Command Unresponsive or slow Auto-heal (request duration) procdump -accepteula -r -dc "Message" -ma Crash (process termination) Crash monitoring Uses DbgHost to capture a memory dump Crash (handled exceptions) Traces in Application Insights/Log Analytics None Crash (unhandled exceptions) Application Insights Snapshot Debugger None Excessive CPU usage Proactive CPU monitoring procdump -accepteula -dc "Message" -ma Excessive memory consumption Auto-heal (memory limit) procdump -accepteula -r -dc "Message" -ma Mapping memory dump scenarios to Azure App Service debugging features This section contains detailed descriptions of the six scenarios that are shown in the previous table. When a request is made to a web server, some code must usually be run. The code execution occurs within the w3wp.exe process on threads. Each thread has a stack that shows what's currently running. An unresponsive scenario can be either permanent (and likely to time out) or slow. Therefore, the unresponsive scenario is one in which a request takes longer than expected to run. What you might consider being slow depends on what the code is doing. For some people, a three-second delay is slow. For others, a 15-second delay is acceptable. Basically, if you see performance metrics that indicate slowness, or a super user states that the server is responding slower than normal, then you have an unresponsive or slow scenario. Over many years, Microsoft .NET Framework has improved the handling of exceptions. In the current version of .NET, the exception handling experience is even better. Historically, if a developer didn't place code snippets within a try-catch block, and an exception was thrown, the process terminated. In that case, an unhandled exception in the developer's code terminated the process. More modern versions of .NET handle some of these “unhandled” exceptions so that the process that's running the code 7 Note We have a secondary recommendation for capturing a W3WP process memory dump in the unresponsive or slow scenario. If that scenario is reproducible, and you want to capture the dump immediately, you can use the Collect a Memory dump diagnostic tool. This tool is located in the Diagnose and solve problems toolset page for the given App Service Web App in the Azure portal. Another location to check for general exceptions and poor performance is on the Application Event Logs page. (You can access Application logs also from the Diagnose and solve problems page.) We discuss all the available methods in the "Expanded Azure App Service debugging feature descriptions" section. Expanded process scenario descriptions Unresponsive or slow scenario Crash (process termination) scenario doesn't crash. However, not all unhandled exceptions are thrown directly from the custom code. For example, access violations (such as 0xC0000005 and 0x80070005) or a stack overflow can terminate the process. Although a software developer takes special care to determine all possible scenarios under which the code runs, something unexpected can occur. The following errors can trigger an exception: Unexpected null values Invalid casting A missing instantiated object It's a best practice to put code execution into try-catch code blocks. If a developer uses these blocks, the code has an opportunity to fail gracefully by specifically managing what follows the unexpected event. A handled exception is an exception that is thrown inside a try block and is caught in the corresponding catch block. In this case, the developer anticipated that an exception could occur and coded an appropriate try-catch block around that section of code. In the catch block, it's useful to capture enough information into a logging source so that the issue can be reproduced and, ultimately, resolved. Exceptions are expensive code paths in terms of performance. Therefore, having many exceptions affects performance. Unhandled exceptions occur when code tries to take an action that it doesn't expect to encounter. As in the Crash (process termination) scenario, that code isn't contained within a try-

catch code block. In this case, the developer didn't anticipate that an exception could occur in that section of code. This scenario differs from the previous two exception scenarios. In the Crash (unhandled exceptions) scenario, the code in question is the code that the developer wrote. It isn't the framework code that's throwing the exception, and it isn't one of the unhandled exceptions that kills the w3wp.exe process. Also, because the code that's throwing an exception isn't within a try-catch block, there's no opportunity to handle the exception gracefully. Troubleshooting the code is initially a bit more complex. Your goal would be to find the exception text, type, and stack that identifies the method that's throwing this unhandled exception. That information enables you to identify where you have to add Crash (handled exceptions) scenario Crash (unhandled exceptions) scenario the try-catch code block. Then, the developer can add the similar logic to log exception details that should exist in the Crash (unhandled exceptions) scenario. What's excessive CPU usage? This situation is dependent on what the code does. In general, if the CPU usage from the w3wp.exe process is 80 percent, then your application is in a critical situation that can cause various symptoms. Some possible symptoms are: Slowness Errors Other undefined behavior Even a 20-percent CPU usage can be considered excessive if the web site is just delivering static HTML files. Post-mortem troubleshooting of an excessive CPU spike by generating a memory dump probably won't help you to determine the specific method that's using it. The best that you can do is to determine which requests were likely taking the longest time, and then try to reproduce the issue by testing the identified method. That procedure assumes that you don't run performance monitors on the performance systems that captured that burst. In many cases, you can cause performance issues by having monitors constantly run in real time. If an application is running in a 32-bit process, excessive memory consumption can be a problem. Even a small amount of activity can consume the 2-3 GB of allocated virtual address space. A 32-bit process can never exceed a total of 4 GB, regardless of the amount of physical memory that's available. A 64-bit process is allocated more memory than a 32-bit process. It's more likely that the 64-bit process will consume the amount of physical memory on the server than that the process will consume its allocated virtual address space. Therefore, what constitutes an excessive memory consumption issue depends on the following factors: Process bitness (32-bit or 64-bit) The amount of memory usage that's considered to be "normal." If your process is consuming more memory than expected, collect a memory dump for analysis to determine what is consuming memory resources. For more information, see Create a memory dump of your App Service when it consumes too much memory . Excessive CPU usage scenario Excessive memory consumption scenario Now that you have a bit more context about the different process scenarios that a memory dump can help you to troubleshoot, we'll discuss the recommended tool for capturing memory dumps on the Azure App Service platform.