Standard Based
AMD is committed to industry standards, offering you a choice in x86 architecture. x86 compatibility means you can run x86 based applications on AMD EPYC instances seamlessly.

Broad Partner Ecosystem
AMD’s broad partner ecosystem and collaborative engineering provide tested and validated solutions that help lower risks associated with new application deployment and application migration.

Cost-effective Cloud Solutions
Amazon Web Services (AWS) cloud instances powered by AMD EPYC processors are priced 10% lower than comparable instances1.

Seamless Workload Migration
x86 compatibility and fully tested software stacks enable migration of applications currently running on other cloud instances to AMD EPYC processor-based instances with little to no modification.

Workload Optimization
AMD engineers have tested a variety of popular applications on Amazon EC2 enabling you to match compute resources to workload needs to optimize cost.

Docker Containers on AMD EPYC Powered AWS EC2 Instances
Docker is a software platform that allows you to build, test, and deploy applications quickly by packaging the software into standardized units called containers that can be run on low cost AMD EPYC processor-based instances provided by AWS offer a low-cost, flexible alternative to on-premise implementations.

AMD EPYC™ Processors: Cost-Effective Computing for Cloud-Based Applications

Increasingly, companies of all sizes are taking advantage of the benefits offered by public cloud providers. The reasons are many and varied: flexible pricing structures, ease of setup, optimization of both staffing and capital budgets, economies of scale, agility, and the ability to go from local to global instantly, are just some of the many benefits.

Leading cloud service providers now feature AMD EPYC™ 7000 series processors to power cloud instances of various types and sizes. The AMD EPYC processor-based instances are fully compatible with all existing x86-based applications. They provide additional options for customers offering a choice for many workloads matching compute resources to application needs at low cost.

AWS EC2 Instances featuring AMD EPYC

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. In partnership with AMD, AWS offers additional EC2 instances that add to their already broad and deep portfolio of instances enabling you to optimize both cost and performance for your workload needs.

EC2 instances featuring AMD EPYC processors provide additional choices to help you optimize both cost and performance for your workloads. They are available in the EC2 general purpose (M5a), general purpose burstable (T3a), and memory optimized (R5a) instance families. EC2 instances featuring AMD EPYC processors deliver a 10% lower cost of compute and memory than comparable instances1. Since many workloads utilize only a fraction of the processor’s maximum performance, these instances offer a better fit for many workloads.

Easily migrate applications currently running on existing EC2 instances to the new AMD EPYC processor-based variants with little to no modification. These instances are available in the same sizes and offer application compatibility with the other x86-based T3, M5, and R5 instances, so you can start using them just like your other EC2 instances. More information is available at https://aws.amazon.com/ec2/amd/.

Figure 1: Linear Scalability of Docker Containers on EPYC processor-powered Instances
Getting Started with Docker Containers on AMD EPYC Processors in the Cloud

All major public cloud providers offer many ways to run containers and containerized applications. AWS offers the Amazon Elastic Container Service (ECS) making it easy to deploy and manage Docker containers in the cloud. When considering container deployment in the cloud, it is important to consider the type of application. Not all applications will benefit from containerization. Containers are good for applications that make use of microservices architectures, because you can link containers together to form a cohesive application, as well as applications that require autoscaling at the operating system level.

One purpose of containerization is to enable the application to automatically scale up and down to meet fluctuating demand by launching and terminating containers and instances as needed. Containers can be rapidly spun up, spun down and duplicated, and can be configured to do so automatically.

This characteristic of containers makes it possible to use computing resources very efficiently. Applications can flexibly spin up containers to meet increases in demand, elegantly providing burst capacity, and just as elegantly spin down containers that are no longer needed. This requires specialized software to manage the containers. As mentioned earlier, AWS offers ECS service for just this purpose.

Another popular container management solution is Kubernetes, commonly referred to as 'K8S.' Kubernetes is open source software available from the Cloud Native Foundation. In addition to ECS, AWS also offers Kubernetes-based container management service called Elastic Container Service for Kubernetes (EKS). AMD EPYC powered AWS instances can be used with either ECS or EKS. The decision to run EKS or ECS depends on a variety of factors outside the scope of this document. For example, ECS is deeply integrated with other AWS services, while EKS may be a better choice in order to integrate with existing Kubernetes-managed containers running on-premise.

Container-based Application Benefits

Containers allow developers to package up an application and all of its components, such as libraries and other dependencies, and deliver it as a single package. Applications are not tied to the host operating system and can be easily moved between local systems and multiple cloud environments.

Most business applications consist of several components organized into stacks, e.g., web server, database, and in-memory cache. Containers make it possible to compose each component into functional units that can be maintained and updated independently.

Finally, and of particular importance when considering container-based applications in the cloud, containers provide the capability to spin up and down compute resources as needed. This allows an organization to both only use computing resources that are actually needed at each point in time, and access burst capacity at times of peak load.

However, care must be taken to choose the appropriate cloud instance type for the workload being processed. It is especially important to understand the nature of the container workload and balance it against the number of CPU cores and vCPUs in the cloud instance (Figure 2).
Selecting the Optimal AMD EPYC processor-powered Instance Type for Docker Container Applications on AWS

Some applications care more about CPU response time, while others are more interested in throughput. Two general recommendations we can make based on our studies are:

1. to achieve optimal performance for CPU response time, balance the number of container instances to the number of physical CPU cores on the virtual machine (VM)
2. to achieve optimal throughput, balance the number of container instances to the number of vCPUs on the VM

Table 1 summarizes the configuration details for the m5a class of instances, and we’ve added an additional column called “CPU.” This column represents the number of physical CPU cores assigned to that VM; in contrast to the vCPU column which represents the number of virtual (or logical) CPUs.

Note that logical cores (vCPUs), also called SMT cores, are virtualized hyper-threading that have been enabled on the CPU so that the VM operating systems see each hyper-thread as a separate (virtual) CPU, hence the name, vCPU. Hyper-threading is an option you can configure when you launch the instance.

Also included is a “Recommendations” column based on our studies to help aid in choosing which instance type is a good starting point for your container application.

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>CPU</th>
<th>vCPUs</th>
<th>RAM</th>
<th>EBS-Optimized Bandwidth</th>
<th>Network Bandwidth</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>m5a.large</td>
<td>1</td>
<td>2</td>
<td>8 GiB</td>
<td>Up to 2.120 Gbps</td>
<td>Up to 10 Gbps</td>
<td>Use with smaller container workloads or microservices.</td>
</tr>
<tr>
<td>m5a.xlarge</td>
<td>2</td>
<td>4</td>
<td>16 GiB</td>
<td>Up to 2.120 Gbps</td>
<td>Up to 10 Gbps</td>
<td>Use with medium size containerized databases or applications.</td>
</tr>
<tr>
<td>m5a.2xlarge</td>
<td>4</td>
<td>8</td>
<td>32 GiB</td>
<td>Up to 2.120 Gbps</td>
<td>Up to 10 Gbps</td>
<td>These are for large size containerized databases or analytics applications. Also, running multiple containers of smaller apps on the same instance.</td>
</tr>
<tr>
<td>m5a.4xlarge</td>
<td>8</td>
<td>16</td>
<td>64 GiB</td>
<td>2.120 Gbps</td>
<td>Up to 10 Gbps</td>
<td>Use the largest instance for running containerized ERP-type workloads or large numbers of containers to be run on the same instance.</td>
</tr>
<tr>
<td>m5a.12xlarge</td>
<td>12</td>
<td>48</td>
<td>192 GiB</td>
<td>5 Gbps</td>
<td>10 Gbps</td>
<td></td>
</tr>
<tr>
<td>m5a.24xlarge</td>
<td>24</td>
<td>96</td>
<td>384 GiB</td>
<td>10 Gbps</td>
<td>20 Gbps</td>
<td></td>
</tr>
</tbody>
</table>

Our studies suggest that for optimal performance with AMD EPYC processor-powered instances, use a 1:1 core to container mapping (SMT=OFF) for applications that are CPU intensive or one that requires rapidly spinning up and down containers (auto-scaling).
Docker Container Studies on AMD EPYC processor-powered Instances

AMD engineers have tested a variety of popular applications on Amazon EC2. This paper gives an overview of the performance testing conducted using Docker Containers.

In order to understand how best to deploy Docker containers on AMD EPYC processor-powered instances on AWS, we ran three studies on two types of instances. Our goal is to help customers make informed decisions about what instance types to choose for their application, as well as understand the impact of different container deployment strategies.

The two instance types we ran our studies on were:

1. m5a.4xlarge (8 CPUs, 16 vCPUs, 64 GiB RAM, 2.120 Gbps EBS bandwidth)
2. m5a.24xlarge (24 CPUs, 96 vCPUs, 384 GiB RAM, 10 Gbps EBS bandwidth)

To characterize system behavior under different workloads we ran three studies:

1. Compute-intensive workload measuring performance and throughput
2. Container density where we run the compute-intensive workload and massively over-provision the system (i.e., start dramatically more containers than is reasonable to identify the stress points in the system)
3. Comparison of three different workloads: compute-intensive (#1 above), compute-intensive with concurrent memory access, and compute-intensive with concurrent memory access and concurrent I/O operations.
Study #1: Compute Intensive Container Workload

For this study, we encapsulated a sysbench CPU test within a container as our compute workload. One sysbench container image will saturate and consume 100% of one CPU. This approach allows us to control the scaling of more homogeneous compute containers and the utilization of CPU and vCPU resources of the instances. To learn more details about the test, please see a similar study that we did on bare-metal.2

Since we’re able to control resource utilization by spinning up more containers, we can make correlations to how a compute intensive workload affects CPU response time and throughput. Figure 3 shows the results of this test on the m5a.24xlarge instance. The left side of the figure shows CPU response time as measured by execution time, while the right side of the figure shows workload throughput as measured by number of events per second.

Figure 3: m5a.24xlarge: Compute Intensive Workload: Performance & Throughput

Looking first at response time, the graph on the left of Figure 3 (lower is better) shows that the compute intensive workload performs optimally all the way up to 100% CPU saturation, meaning there is virtually no performance penalty for running at high CPU usage levels. We can also clearly see the point at which another VM needs to be spun up if response time is the critical factor for the application.
Study #1: Compute Intensive Container Workload (cont.)

It’s important to note that in this study we are artificially consuming large amounts of CPU resources without regard for other aspects of the system like memory or I/O. A real-world application will have a different CPU usage profile that is likely not stressing the system as severely as we are here. The key takeaway is worth repeating: there is virtually no performance penalty for running at very high CPU usage levels.

Turning our attention to throughput, the graph on the right of Figure 3 (higher is better) tells a similar story. The throughput increases in a linear fashion as containers are added up to the point where the virtual CPU is completely saturated. We can make the same statement here: the system behaves in a very predictable way, and there is a clear point where another VM needs to be spun up if throughput is the key factor for the application.

![Graph of Compute Workload Response Time for m5a.4xlarge vs. Number of Concurrent Containers](image1)

![Graph of Compute Workload Throughput of m5a.4xlarge VMs vs. Number of Concurrent Containers](image2)

**Figure 4: m5a.4xlarge: Compute Intensive Workload: Performance & Throughput**

Figure 4 shows the same tests run on the smaller m5a.4xlarge instance. The results are similar with the system behaving in a predictable, linear fashion up to the point of 100% CPU saturation.

One other item worth noting is that the throughput of the application can be further boosted by saturating the vCPUs (logical cores) of the instance. Note the EPYC-SMT line on the right side of Figures 3 and 4. This requires enabling hyper-threading (SMT) so the vCPUs appear to the operating system. However, not all applications will benefit from hyper-threading. vCPUs share resources, like L3 cache and memory, among themselves. A compute intensive workload that, for example, consumes a large portion of the L3 cache, may not provide a significant throughput boost.

Whether to scale up compute intensive container workloads vertically on a single large instance or horizontally across smaller instances depends on other considerations. For microservices that are latency sensitive and have dependencies on other services, deploying these containers on the same instance can help to reduce the latency issue, but introduces a single point of failure. If the application has a high-availability requirement, then scaling horizontally across multiple smaller instances may prove to be a better choice.
Study #2: Container Density

In some cases, compute density, i.e., spinning up as many containers as possible, is more important than either performance as measured by CPU response time or application throughput. To study compute density, we looked at the behavior of an instance as we add containers running the compute-intensive workload from our first study. Specifically, we took the number vCPUs needed to completely saturate the CPU, i.e., 100% CPU utilization, and instantiated ten times that number of containers. Figure 5 shows that the m5a.24xlarge instance can process ten times the load of full vCPU saturation with predictable, linear CPU response times, but throughput levels out, also in a predictable way.

![Figure 5: Container Density of m5a.24x Instance with Compute Intensive Workload](image)

We repeated the same compute density testing on the smaller m5a.4x instance. In this overprovisioning scenario, the behavior of the smaller instance, shown in Figure 6, is similar to that of the larger m5a.24xlarge instance. In both cases, compute intensive load scales linearly. For applications where performance is not a priority but cost is a bigger factor, scaling compute-intensive workloads on large and small EPYC processor-powered instances behave in a very predictable way.

![Figure 6: Container Density of m5a.4x Instance with Compute Intensive Workload](image)
Study #3: Comparison of Different Workloads

For our final study, we wanted to understand how a heavy combination of workloads affect CPU response time and throughput of both the m5a.24x and m5a.4x instances. We created combined workloads where the system is processing memory and I/O intensive tasks concurrent with the compute intensive tasks from our first studies. To that end we created two new workloads. We took the compute-intensive workload from the first two studies and we added:

- A memory workload, with the average memory utilization set at 75%, and
- A memory plus I/O workload, with memory utilization still at 75% and I/O utilization set at ~99%

![Figure 7: Comparison of Different Workloads on m5a.24xlarge Instance](image)

Figure 7 shows how these combined workloads compare to the compute intensive workload on the larger m5a.24x instance. The key take-away from these results is that at the higher CPU utilization rates, CPU performance during the combined workload tracks closely to the performance when running only the compute-intensive workload. This highlights that EPYC-processor powered instances can handle heavy I/O and memory tasks even at the higher end of CPU utilization (>80%). This provides you with an option to do more work on EPYC processor-powered instances without spinning up another VM to offload the work.
Study #3: Comparison of Different Workloads (cont.)

We ran the same tests on the smaller m5a.4x instance. The results are interesting, and are shown in Figures 8 and 9. For this smaller instance, the system performs as expected for the combined workloads, but only up to ~66% CPU utilization (Figure 8).

![Figure 8: Comparison of Different Workloads on m5a.4xlarge Instance up to ~66% CPU utilization](image)

Beyond this utilization rate, the smaller instance has less to work with and we start to see thrashing that leads to unpredictable behavior. Application throughput and CPU response time for the combined workloads becomes unpredictable and does not scale linearly (Figure 9).

Our conclusion from this test is that a compute intensive container application with concurrent heavy I/O and memory intensive tasks should not scale above 66% vCPU utilization on the smaller instance. Spin up a second VM if additional resources are needed.

![Figure 9: Comparison of Different Workloads on m5a.4xlarge Instance – Over-provisioned](image)
Further Reading

For more information visit the links below:

- Amazon Web Service’s instances powered by AMD EPYC processors: [https://aws.amazon.com/ec2/amd/](https://aws.amazon.com/ec2/amd/)
- Kubernetes for container management: [https://kubernetes.io/docs/concepts/](https://kubernetes.io/docs/concepts/)
Conclusion

Incorporating AMD technology, AWS offers additional EPYC processor-powered EC2 instances that add to their already broad and deep portfolio of instances enabling you to optimize both cost and performance for your workload needs.

AMD’s broad partner ecosystem and collaborative engineering provide tested and validated solutions that help lower risks associated with new application deployment and application migration in container environments.

Footnotes


DISCLAIMER

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale. GD-18

©2019 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, EPYC, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Docker is a registered trademark of Docker, Inc. in the United States and/or other countries. Amazon Web Services, and the “Powered by AWS” logo are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.