

[Back to Overview](#)

Running a custom SQL Server Docker Image in Amazon AWS

November 26, 2018 • Klaus Aschenbrenner • Docker, Linux, SQLServer • One Comment

In one of my [last blog postings](#) I have shown you how I run SQL Server natively on my Mac. In today's blog posting I want to continue with that approach, and I want to show you how you can create a custom SQL Server Docker Image, and how you can run that one in Amazon ECS - Amazon Container Services.

Creating and publishing a custom SQL Server Docker Image

If you have followed my blog posting about how to [run SQL Server on the Mac](#), you should have now a fully working SQL Server 2019 CTP 2.1 environment running locally. One of the greatest things about Docker Images is also that you can create your own custom images that you can finally publish for other peers for reuse.

If you create your own custom image, your image will be based on some other image. Docker uses internally a layered file system, and therefore your custom image will be also quite small, because it only contains the changes on which your custom image is based on. In our case, the custom image that we will build today, is based on the SQL Server 2019 CTP 2.1 image.

One of the first things that I'm always doing when I install SQL Server is the restore of some databases that I will always use. In my case that's the **AdventureWorks** database, and in some cases I also restore the **ContosoRetailDW** database. You have already seen in my last blog posting how you can copy a database backup file into your SQL Server container, and how you can perform the restore itself.

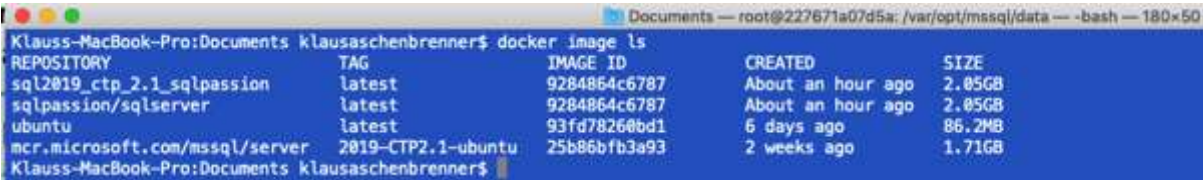
To make things easy today (and the custom Docker image small), I have only restored the **AdventureWorks** database. After the restore operation I have finally stopped the Docker Container with the following command line:

```
docker stop sql2019_ctp2.1
```

As soon as your Docker Container is in a stopped state, you can create your custom Docker Image that is based on the current state of the chosen Docker Container. To create a custom Docker Image you will use the following command line:

```
docker commit sql2019_ctp2.1 sql2019_ctp2.1_sqlpassion
```

As you can see from the command line, you have to pass in the name of the Docker Container, and the name of your new custom Docker Image. Easy, isn't it? If you now check your local Docker Image Repository, you can see that you have an additional Docker Image available - the custom one that you have just built!



Publishing a custom Docker Image

But how can you now publish your local created Docker image for yourself and also for other people? The answer is quite easy: you create your own repository on docker.com! If you check-out the following URL, you can see my own repository that I have created:

https://hub.docker.com/r/sqlpassion/

To be able to push custom Docker Images into your repository, you have to tag them accordingly. In my case I have tagged the image with the sqlpassion/sqlserver:latest tag. If you don't tag your image, you can't push it into your repository!

```
docker image tag sql2019_ctp_2.1_sqlpassion:latest sqlpassion/sqlserver:latest
```

And after the tagging is done, let's push the image into my Docker repository:

```
docker push sqlpassion/sqlserver
```



If you search on your local Docker installation for **sqlpassion**, you will be also able to find my custom Docker Image:



If you pull that one, you can deploy it locally on your hardware. Docker is just cool! 😊

Running a Docker Container in Amazon AWS

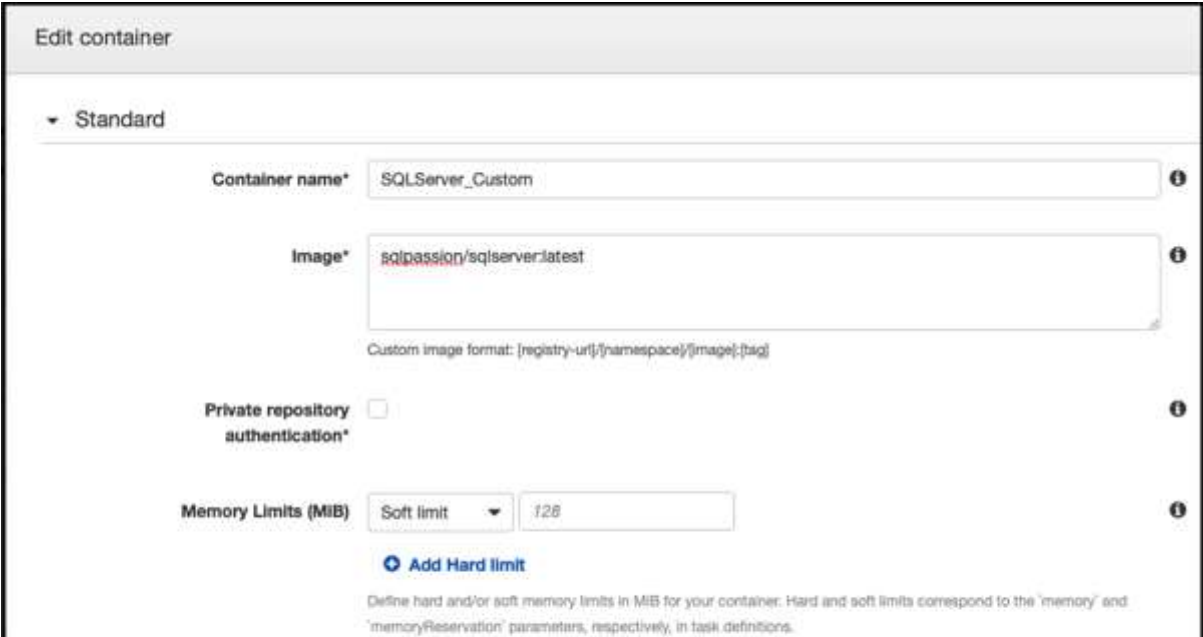
In my case I wanted to try how I can run a custom Docker Image in the cloud. Because I'm using the Amazon AWS, and especially Amazon EC2 already for some years, I wanted to try if it is possible to spin up my custom Docker Image in Amazon AWS.

One of the cool things that I have found out about Amazon AWS is the fact that they offer Amazon ECS – Amazon Container Services. You can run natively a Docker Container in Amazon AWS without creating EC2 instances. The creation of the necessary Virtual Machines (that actually run your Docker Containers) is just done transparently in the background of Amazon ECS.

So let's have now a look on the necessary steps to spin up a few instances of my custom Docker Image. To be able to run a Docker Container, you have to configure the following Amazon ECS objects:

- Container Definition
- Task Definition
- Service Definition
- Cluster Definition

So let's have a look on these objects, and how to configure them accordingly. In the first step you have to create a **Container Definition**. You can choose from existing Container Images (like nginx, tomcat, ...), or you choose a third-party Docker Image – like the one we have created in the previous step.



ECS recommends 300-500 MiB as a starting point for web applications.

Port mappings	Container port	Protocol
	1433	tcp

[Add port mapping](#)

As you can see, you give the Container a name, and you have to specify your Docker Image source. When you don't specify a Registry URL, they are defaulting to the Docker Registry. In my case I have provided the Image URL sqlpassion/sqlserver:latest. And you also have to create a Port Mapping for the port 1433, so that we can access SQL Server from the outside. And under the Advanced Container Settings, I have specified the necessary environment variables for SQL Server:

You may also designate AWS Systems Manager Parameter Store keys or ARNs using the 'valueFrom' field. ECS will inject the value into containers at run-time.

Key	Value	Value
ACCEPT_EULA	Value	Y
SA_PASSWORD	Value	password1!

In the next step you have to configure the Task Definition. With a **Task Definition** you specify how many CPUs and how much memory you want to assign to your Docker Container. In my case I have given the Docker Container 4 CPUs and 8 GB RAM.

Task size

Task size allows you to size at the task level and optionally set container-specific CPU and memory sizes. You are billed for the task memory and task CPU allocated.

Task memory* 8GB (8192)

Task CPU* 4 vCPU (4096)

With the **Service Definition** you can specify how many instances of your Docker Container you want to start. In my case I have configured just 1 Task.

Define your service [Edit](#)

A service allows you to run and maintain a specified number (the "desired count") of simultaneous instances of a task definition in an ECS cluster.

Service name **SQLServer_Custom-service**

Number of desired tasks **1**

Security group **Automatically create new**
A security group is created to allow all public traffic to your service only on the container port specified. You can further configure security groups and network access outside of this wizard.

Load balancer type ☒ None ☐ Application Load Balancer

And finally you have your **Cluster** a name:

Configure your cluster

The infrastructure in a Fargate cluster is fully managed by AWS. Your containers run without you managing and configuring individual Amazon EC2 instances.

To see key differences between Fargate and standard ECS clusters, see the [Amazon ECS documentation](#).

Cluster name **SQLpassionCluster**
Cluster names are unique per account per region. Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

VPC ID **Automatically create new**

Subnets **Automatically create new**

After you have confirmed all your settings, it takes a few minutes and your Service is up and ready. When you look into the tab Logs, you can see that SQL Server is up and running within the Docker Container:

Clusters > SQLpassionCluster > Service: SQLServer_Custom-service

Service : SQLServer_Custom-service

ClusterSQLpassionCluster

StatusACTIVE

Task definitionfirst-run-task-definition:6

Service typeREPLICA

Launch typeFARGATE

Platform versionLATEST(1.2.0)

Service roleAWSServiceRoleForECS

Desired count1

Pending count0

Running count1

DetailsTasksEventsAuto ScalingDeploymentsMetricsTagsLogs

Task statusRUNNINGSTOPPED

Filter logs

X

All30s5m1h6h1d1w

Timestamp (UTC+00:00)	Message	Task
2018-11-26 14:15:14	2018-11-26 13:15:14.33 spid9s Recovery is complete. This is an informational message only. No user acti...	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:14	2018-11-26 13:15:14.32 spid26s Service Broker manager has started.	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:14	2018-11-26 13:15:14.31 spid26s The Database Mirroring endpoint is in disabled or stopped state.	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:14	2018-11-26 13:15:14.31 spid26s The Service Broker endpoint is in disabled or stopped state.	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:14	2018-11-26 13:15:14.30 spid12s The tempdb database has 1 data file(s).	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:14	2018-11-26 13:15:14.16 Server Enclave of type 0 initialized successfully.	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:14	2018-11-26 13:15:14.13 spid12s Starting up database 'tempdb'.	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:14	2018-11-26 13:15:14.13 spid12s [2]. Feature Status: PVS: 0, CTR: 0, ConcurrentPFUpdate: 1.	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:13	2018-11-26 13:15:13.88 Server Failed to verify the Authenticode signature of 'C:\bin\sectorwarder.dll'. Si...	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:13	2018-11-26 13:15:13.75 Server Common language runtime (CLR) functionality initialized.	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3
2018-11-26 14:15:13	2018-11-26 13:15:13.73 spid29s Parallel redo is shutdown for database 'AdventureWorks2014' with work...	34cb8e23-6e97-450a-8f79-8d1ce32ac7b3

If you go to the Task itself, you can see in the Networking settings the public IP of your Docker Container. You can use that IP address within SQL Server Management Studio or Azure Data Studio to connect to your SQL Server Instance:

Clusters > SQLpassionCluster > Task: 34cb8e23-6e97-450a-8f79-8d1ce32ac7b3

Task : 34cb8e23-6e97-450a-8f79-8d1ce32ac7b3

DetailsTagsLogs

ClusterSQLpassionCluster

Launch typeFARGATE

Platform version1.2.0

Task definitionfirst-run-task-definition:6

Groupservice:SQLServer_Custom-service

Task roleNone

Last statusRUNNING

Desired statusRUNNING

Created at2018-11-26 14:13:28 +0100

Network

Network modeawsvpc

ENI Ideni-7e063a29

Subnet Idsubnet-043f1dec02fb48d2e

Private IP

Public IP

Mac address02:ee:3d:ef:12:cc

If you have connected to SQL Server, you can finally see that you have deployed a custom Docker Image, because the **AdventureWorks** database is already there!

Summary

I have done over the last few weeks a lot of research and work with Docker and Kubernetes. But Docker changes everything! Imagine what you can do with custom published Docker Images. You could have for each conference talk a custom Docker Image, and the attendees are just pulling it from your Docker Repository to follow along your talk.

Or imagine you want to offer Lab-based training materials. You could create a custom Docker Image for some Hands-On Labs that you can run on your local hardware, or in the Cloud. The options are just endless – at least for me...

Thanks for your time,
-Klaus

1 thought on “Running a custom SQL Server Docker Image in Amazon AWS”



SHANJAMES
11/08/2019 AT 11:10 AM

Thanks for sharing this article as its a great example of how easy it is to suddenly find yourself in trouble in these environments.

[Reply](#)

Leave a Comment

Your email address will not be published. Required fields are marked *

Type here..

Name*

Email*

☐ Save my name, email, and website in this browser for the next time I comment.

Website

Post Comment »



SQLpassion Performance Tuning Training Plan – Signup

Be sure to checkout the [FREE SQLpassion Performance Tuning Training Plan](#) – you get a weekly email packed with all the essential knowledge you need to know about performance tuning on SQL Server.

Your E-Mail

