

Docker Toolkit for Data Scientists – How to Start Doing Data Science in Minutes!

Donna De Capite. SAS Institute Inc.

ABSTRACT

SAS® continues to grow its capabilities of running SAS in the cloud with containers! Learn what a container is and how it can be used to run SAS® Analytics for Containers and leverage the power of in-memory compute capabilities with SAS® Viya®. This paper discusses how SAS containers run in a variety of cloud platforms including Amazon Web Services, Google Cloud Platform, Microsoft Azure, as well as how they run on a private OpenStack cloud using Red Hat OpenShift. Additional topics include provisioning web-browser based clients via Jupyter Notebooks and SAS® Studio to empower data scientists with the tool of their choice, and how to take advantage of working with SAS Analytics for Containers with Hadoop.

INTRODUCTION

This paper introduces the concept of a container, discusses why there is a growing trend to run software in a container, and shows how to run the SAS Analytics for Containers in a variety of public and private clouds.

An important skill to have when working with containers is knowledge of the Linux operating system. Many of the interactions with containers require working with the Linux command line. This paper presents the requirements for getting containers into the necessary environment. In Figure 1 below, the user interacts with SAS through their browser. The browser connects to a URL that represents the address of the SAS process running in the cloud. The SAS process is a Docker container that could be running in either a public or a private cloud. You could run the container on your laptop or desktop and connect to the container from your browser, but this paper is focused on the deployments of SAS containers in the cloud.

Figure 1 is a high-level overview diagram of how you interact with a SAS container. The user interacts with a container through a browser running on their laptop or other device. The container is actually running in the cloud, where it is taking advantage of the compute resources and infrastructure that are offered by the cloud. Docker is also in the cloud.

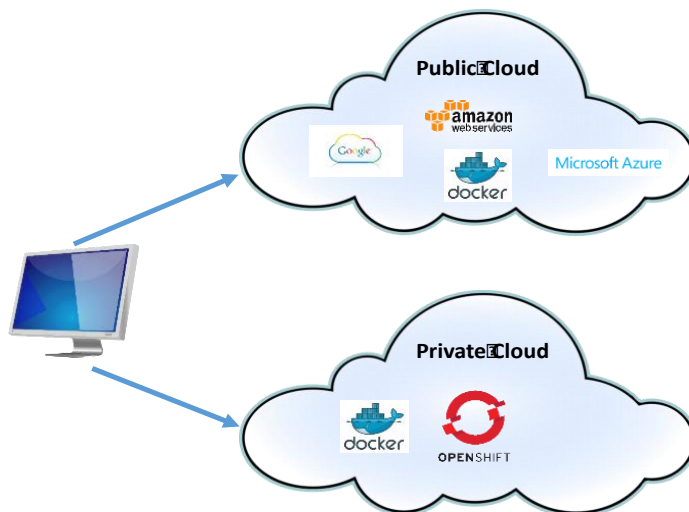


Figure 1. High-level Overview of Interacting with a Container

CONTAINER BASICS

This section introduces Docker and some of the basic concepts of working with Docker containers. In 2017, I wrote a paper on the SAS Docker container. Since that time, I have made some changes in the Dockerfile, and those changes are included in this document. See the Recommended Reading section of this paper for the reference to this paper.

The best way to describe a container is to compare it to a shipping container. Shipping containers are used all over the world to transport goods from one place to another. The contents of the container are loaded and the container is picked up and moved to its destination. Along the way, it can be moved around easily. All the containers look the same to the equipment moving the container, whether that equipment is a forklift, ship, train, or tractor trailer truck. The contents are completely unique to each container. A software container is like a shipping container. It has a standard format that is based on open-source standards, and each container is defined specifically to run the software. A software container holds the code executables, as well as any libraries, system tools, and anything else that is needed to run the software. The container also describes the operating system on which the executables. Most containers use Linux. CentOS is a popular distribution of Linux, because it is an upstream open-source version of Red Hat Enterprise Linux (RHEL). If you have ever downloaded software and then needed to run additional updates or hunt down a software or library dependency, you can understand how the container can ease your pain. The container has in it all the pieces that are needed to run your software. It eliminates the need to have different configurations to cover conflicting dependencies across all the software that is installed on a system. The software container described in this paper is a Docker container, but similar principles can be applied to other container providers. The container can be built once and can be run anywhere that runs containers.

CONTAINER VERSUS A VIRTUAL MACHINE

Many people are familiar with working with virtual machines, and the container concept is very similar. A virtual machine also has all of the necessary pieces to run software. However, the way that the virtual machine consumes resources is very different from that of the container. The biggest difference that is visible to end users is that the container shares the operating system and Linux kernel with other containers that are running on the same physical host. The operating system is already running for the containers, which allows for quicker start-up time. A virtual machine needs to boot up the operating system each time it starts up. Another advantage of using containers is that the container image is registered in a central repository. This allows the container to be referenced and easily deployed from that image. After you pull down the container from the repository start it up, the running software is then available.

WHAT IS A CONTAINER ORCHESTRATOR?

Our Docker container is executing in our cloud environment and doing all the work for our requests. So, if the Docker container is the **WHAT**, then the container orchestrator is the **HOW, WHEN, and WHERE**. The orchestrator tells the containers how to run and allows for scaling up and down. It can provide the run-time settings for the container. The orchestrator can also provide settings about the resources that are needed to run that particular container, so you can tune these settings based on the expected workload. The orchestrators take care of making sure that the minimum settings needed to run the container are met when provisioning resources for the container. The orchestrators can also provide some amount of scheduling about when containers start and stop. Because of these factors, the orchestrators are a major part of any enterprise deployment of containers. The container itself only holds the pieces that are needed to run. The orchestrator can add variables to fine-tune how the container runs. There are many different container orchestrators.

Figure 2 below illustrates several of the most used container orchestrators. After you have decided to work with Docker containers, your next step in managing how the containers will run under your corporate workloads will likely be to choose one of these orchestrators. Although some customers have “rolled their own” method for managing the complexities involved with containerized workloads, more and more customers have adopted orchestrators. Kubernetes is currently the market leader.



Copyright © SAS Institute Inc. All rights reserved.



Figure 2 Container Orchestrators

KUBERNETES - The Google-designed Kubernetes is an open-source system for Docker container management and orchestration.

AMAZON EC2 CONTAINER SERVICE (ECS) - Amazon Elastic Compute Cloud (EC2) Container Service is a container management service for Docker containers. Importantly, any containers managed by Amazon ECS will be run only on instances of Amazon Web Services EC2; so far, there is no support for external infrastructure.

MARATHON FOR MESOS AND DC/OS - Marathon is a production-grade open-source framework for container management and orchestration that is based on [Apache Mesos](#) and is intended to work with applications or services that will run over a long period of time.

DOCKER SWARM - Docker’s own tool for cluster management and orchestration was introduced into Docker Engine as “swarm mode” with the Docker 1.12 update. This update added support to the Docker Engine for multi-host and multi-container orchestration. Note: During the time that this paper was being prepared, Docker announced support for Kubernetes as well; see <https://www.docker.com/kubernetes> for more details.

AZURE CONTAINER SERVICE (ACS) – The container orchestration solution provided by Microsoft for its Azure cloud computing platform, [Azure Container Service](#), reached general availability in April 2016. ACS is based on the open-source Apache Mesos cluster manager, and enables users to choose between three container orchestration tools: Apache Mesos, Docker Swarm, and Kubernetes.

A container orchestrator should be in place for any enterprise deployment of containers. An organization that has initiated moving to containerized workloads should choose some type of orchestration to help manage their containers. The portability of the Docker container makes a container fit into any of these orchestrators because they all work with the Docker standard.

BUILDING A SAS CONTAINER

One of the first containers that SAS has made available to customers is the SAS Analytics for Containers. This container is built using a tarball of the SAS Linux deployment, which is then referenced in the container. Figure 3 is an example of building a Docker container. The process starts with laying down CentOS Linux, and then adding the necessary libraries to run SAS. We then add a group and user to configure the container's Linux environment the way we want it to be on the file system. This example includes a VERY rough way of adding a password and user to the container). We create the SASHome directory, add the pieces of SAS to this SASHome directory, and set the proper permissions on all the files in SASHome. We expose a port so that the container can communicate out to our web browser. The addition of the entry.sh allows us to provide a script that will always run each time the container is started.

Figure 3 is an example Dockerfile. This Dockerfile is the basic template to start and run a SAS container. A typical production Dockerfile could also have mount points for customizations such as persistent storage, user authentication, and SAS settings, including formats and pre-assigned LIBNAME statements.

```
FROM centos
MAINTAINER Your Name
## install necessary libraries
RUN yum -y install numactl-libs.x86_64 libXp passwd libpng12 libXmu.x86_64
RUN groupadd -g 1001 sas
RUN useradd -m -u 501 -g sas sas
## example set default password for this example only
RUN echo -e "yadayada" | /usr/bin/passwd --stdin sas
RUN mkdir -p /usr/local/SASHome
ADD SAS.tar /
RUN chown -R sas:sas /usr/local/SASHome
EXPOSE 38080
ADD entry.sh /
RUN chmod +x /entry.sh
ENTRYPOINT ["/entry.sh"]
```

Figure 3. Example Dockerfile

Figure 4 is an example entry.sh script. This script runs each time the container is started. It starts SAS Studio so that a web browser can connect to SAS through SAS Studio.

```
#!/bin/bash
./usr/local/SASHome/SASFoundation/9.4/utilities/bin/setuid.sh
./usr/local/SASHome/sas/studioconfig/sasstudio.sh start
tail -f /dev/null
```

Figure 4: Example entry.sh

DEPLOYING A SAS CONTAINER

Just to clarify some terminology a bit, it can be overwhelming for people just starting to work with the “cloud” to understand what we mean by the cloud. When we think of the cloud, it could mean running workloads in our own company's data center, where the compute and infrastructure is managed by our company's Information Technology (IT) folks. This is known as a private cloud or on-premise. Another interpretation of running in the “cloud” could be running processes on a public cloud, where the compute and infrastructure are provisioned and managed by a public cloud provider such as Amazon Web

Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, or Oracle Cloud,. If you use a public cloud provider and need additional resources to run your workload, you don't have to find or purchase more hardware to run the workload as you would with on-premise data centers. Instead, you just get what you need out of your public cloud provider. Because more companies are finding it more cost effective and easier to get what they need from a public cloud, more customer data and processing has moved to the public cloud providers.

With each passing year, the public cloud providers have made it easier to deploy workloads to their clouds. This paper looks at deploying to AWS, GCP, and Azure public clouds and Red Hat Openshift for private cloud.

It is well known that Docker is a standard where you build it once and it essentially can run anywhere that supports Docker. However, each cloud provider has a different way to get the container deployed. The following sections will outline the steps I went through to get SAS Docker containers to run in cloud.

Please note, there are many ways to run containers in cloud environments. One of the more basic ways of running containers is to provision a Linux machine in your cloud environment, install Docker, and deploy your container. For the purpose of this paper, I chose a container orchestrator for each cloud and provided many screen shots to show how I deployed to each cloud.

PREREQUISITES

Each cloud provider has a command line tool that you need to download to your PC. For all the screenshots provided, there is a way to perform each step completely using the command line interface (CLI). This paper illustrates the process using the user interface (UI) as much as possible, and uses the CLI only when absolutely necessary. Use these locations to download the command line tool for each provider.

Google Cloud Platform	https://cloud.google.com/sdk/
AWS	https://aws.amazon.com/cli/
Azure latest	https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest
Openshift	https://docs.openshift.com/enterprise/3.1/cli_reference/get_started_cli.html

The Docker images are built on a Linux environment. I installed Docker on Linux and pulled all the CLIs needed to run with the various cloud environments. In my Linux environment, a Docker repository held my Docker images, and these are the images that I pushed to the various cloud environments. In all cases I needed to authenticate to the cloud provider from my Linux environment.

AWS – ELASTIC CONTAINER SERVICE

The first example in deploying to a public cloud uses the AWS Elastic Container Service. ECS is one of Amazon's offerings to easily deploy and manage Docker containers. ECS provides a cluster of machines that can be used to deploy containers. It provides features like security groups, volumes, load-balancing, and the ability to work with virtual private connections (VPCs).

AWS offers quick start capability to enable you to start using ECS. The first step is to create a repository that will hold your Docker container image.

Step 1: Configure Amazon Elastic Container Service (ECS) Repository

The repository holds the Docker containers that will run in the service. Typically, developers build their Docker containers in their own playpen and then push them to a production container repository. The Docker container must be pushed to the ECS repository. Figure 5 shows how to configure AWS repository.

Getting Started with Amazon Elastic Container Service (ECS)

Step 1: Configure repository

Step 2: Build, tag, and push Docker image

Step 3: Create a task definition

Step 4: Configure service

Step 5: Configure cluster

Step 6: Review

Configure repository

The wizard guides you through creating a repository in Amazon ECR. [Learn more](#)

Repository name* ⓘ

Namespaces are optional, and they can be included in the repository name with a slash (for example, namespace/repo)
Repository name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, and forward slashes.

Repository URI 984156722984.dkr.ecr.us-east-2.amazonaws.com/sa4c_r

Permissions

As the owner, you have access to this repository by default. After completing this wizard, you can grant others permission to access this repository in the console.

*Required Cancel Previous Next step

Figure 5: AWS - Configure Repository

Step 2: Tag and Push Docker Image

The quick-start steps provide the name of the location to which you will push the container image. Working in a Linux command line terminal where your local Docker repository is accessible, tag the local image to the location in AWS ECS. The next step is to push the image to ECS. Figure 6 shows the steps to tag and push to the Amazon container repository.

```
$ docker tag sa4c:latest 984156722984.dkr.ecr.us-east-2.amazonaws.com/sa4c_r:latest
$ docker push 984156722984.dkr.ecr.us-east-2.amazonaws.com/sa4c_r:latest
The push refers to a repository [984156722984.dkr.ecr.us-east-2.amazonaws.com/sa4c_r]
462bfe8e4b3b: Pushed
ac0c84bbab7e: Pushed
85a4fc2bed5f: Pushed
677cbf8e0c66: Pushed
a45efa5acce: Pushed
d90058967009: Pushed
37af538e9ab0: Pushed
b072bd4263ea: Pushed
1f49ca18c653: Pushed
f4ddd8e8095: Pushed
36018b5e9787: Pushed
latest: digest: sha256:08fa27400309d542eac0ee5f0b680ae52b9bff2e55bdbf77327a433bf481a0d2 size: 2619
$
```

Figure 6: AWS - Tag and Push to Repository

After these steps, the SAS container image is registered in the Amazon repository.

Step 3: Create a Task Definition

For the purpose of demonstrating how to integrate with AWS, this example uses FARGATE, because it takes away the need to manage EC2 instances. Figure 7 shows the main screen when creating a new Task Definition.

Create new Task Definition

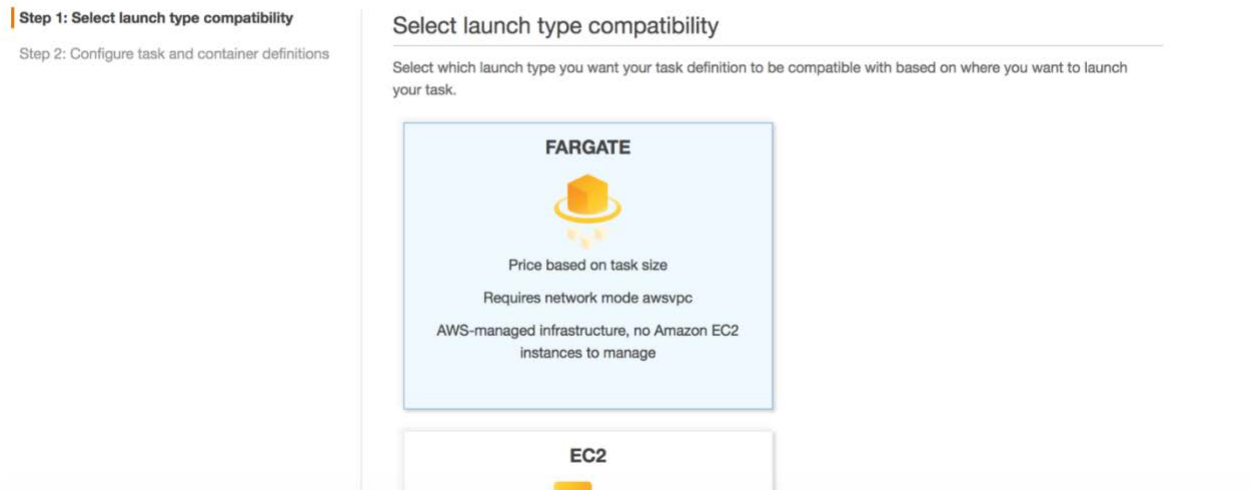


Figure 7: AWS - Create FARGATE Task Definition

The task is the definition on how to run the container. The task sets the amount of memory needed and the number of virtual CPUs. This setting should reflect the type of workload expected to be performed by the container. It is easy to redeploy the container with more or fewer resources. Figure 8 illustrates that the task size for running the SAS container allocates 10 GB of memory and four virtual CPUs. These settings depend on the workload planned for the SAS container.

Task size ?

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (GB)

The valid memory range for 4 vCPU is: 8GB - 30GB.

Task CPU (vCPU)

The valid CPU range for 10GB memory is: 2 vCPU - 4 vCPU.

Task memory maximum allocation for container memory reservation

0 9984 shared of 10240 MiB

Task CPU maximum allocation for containers

0 4096 shared of 4096 CPU units

Container Definitions ?

[Add container](#)

Figure 8: AWS - Task Settings

Step 4: Configure a Service

When configuring a service, configure the network with a VPC/subnet, create or use an existing security group, and set the optional auto scaling feature. Figure 9 shows a summary screen of the settings that are used to configure a service.

Create Service

- Step 1: Configure service
- Step 2: Configure network
- Step 3: Set Auto Scaling (optional)
- Step 4: Review**

Review

Edit

Cluster DD-Cluster
Launch type FARGATE
Task Definition new-task:1
Service name SA4C
Number of tasks 1
Minimum healthy percent 50
Maximum percent 200

Configure network

Edit

VPC Id vpc-716e3909
Subnets subnet-181fa245
Create new security group SA4C-6822
Auto assign IP ENABLED

Set Auto Scaling (optional)

Edit

not configured

Cancel Previous **Create Service**

Figure 9: AWS - Configure Service Summary

After creating the service definition, the process to configure the service is launched. Figure 10 shows the steps involved with configuring the service, including creating a security group and setting inbound rules.

Launch Status

ECS Service status - 3 of 3 completed

Configure Task Networking

Create security group

Create security group

SA4C-6822 succeeded sg-913075e6 (<https://us-east-1.console.aws.amazon.com/ec2/v2/home?region=us-east-1#SecurityGroups:groupId=sg-913075e6>)

Set inbound rules

Set inbound rules

succeeded sg-913075e6 (<https://us-east-1.console.aws.amazon.com/ec2/v2/home?region=us-east-1#SecurityGroups:groupId=sg-913075e6>)

Create Service

Create service: SA4C

Service created

Service created. Tasks will start momentarily. [View: SA4C](#)

Additional integrations you can connect to your ECS service

Code Pipeline

Setup a CI/CD process from your service. You can build from source or have an ECR repository as the source for your deployment.

Create a pipeline (<https://us-east-1.console.aws.amazon.com/codepipeline/home?region=us-east-1#/create/Name>)

[Back](#)

[View Service](#)

Figure 10: AWS - Create Service Launch Status

Step 5: Configure a Cluster

The final step is to configure the cluster where the containers will run. Figure 11 shows the launch status of creating the DD-Cluster below.

Launch status

Your container instances are launching, and it may take a few minutes until they are in the running state and ready to access. Usage hours on your new container instances start immediately and continue to accrue until you stop or terminate them.

[Back](#) [View Cluster](#)

ECS status - 1 of 1 complete **DD-Cluster**

✔ **ECS cluster**
ECS Cluster DD-Cluster successfully created

* Required

[Cancel](#)

[Previous](#)

[Create](#)

Figure 11: AWS - Configure Cluster Launch

If you look at the task definition in Figure 12, it shows that the DD-Cluster is running a FARGATE template. The associated ENI is eni-998cb546. If you navigate to the ENI and select it, you can see that the associated security group for this ID is SA4C-6822. The security group defines what traffic can communicate with the container, as well as inbound and outbound rules. The SAS container in this example requires that port 38080 be open so that it can be connected to through your browser. Figure 12 also shows the public IP address for the container. Assuming that all of the networking is open for port 38080, to navigate to the container, you only have to find the public IP and tag the port. You then will be launched into SAS Studio that is running in a container on AWS. This URL is an example.

<http://54.161.27.146:38080/>

- Amazon ECS
- Clusters**
- Task Definitions
- Repositories

AWS recently announced Fargate (https://aws.amazon.com/about-aws/whats-new/2017/11/introducing-aws-fargate-a-technology-to-run-containers-without-managing-infrastructure/), a new ECS launch type currently available in the us-east-1 (N. Virginia) Region. Fargate allows you to run tasks and services on infrastructure managed by AWS so that you no longer have to manage your own EC2 instances. Get started (https://console.aws.amazon.com/ecs/home?region=us-east-1#/firstRun) Learn more (https://aws.amazon.com/fargate/)

Clusters > DD-Cluster > Task: be7793e7-a5af-4de5-9978-37e004ace026

Task : be7793e7-a5af-4de5-9978-37e004ace026

Details Logs Run more like this Stop

Cluster	DD-Cluster
Launch type	FARGATE
Platform version	1.0.0
Task definition	new-task:1
Group	service:SA4C
Task role	ALLTHINGS (https://us-east-1.console.aws.amazon.com/iam/home?#roles/ALLTHINGS)
Last status	RUNNING
Desired status	RUNNING
Created at	2018-02-13 15:36:35 -0500

Network

Network mode	awsvpc
ENI Id	eni-998cb546 (https://us-east-1.console.aws.amazon.com/ec2/v2/home?region=us-east-1#NIC:networkInterfaceId=eni-998cb546;sort=desc:networkInterfaceId)
Subnet Id	subnet-181fa245
Private IP	10.0.1.214
Public IP	54.161.27.146
Mac address	0e:89:d1:00:10:28

Containers

Last updated on February 13, 2018 3:55:52 PM (4m ago)

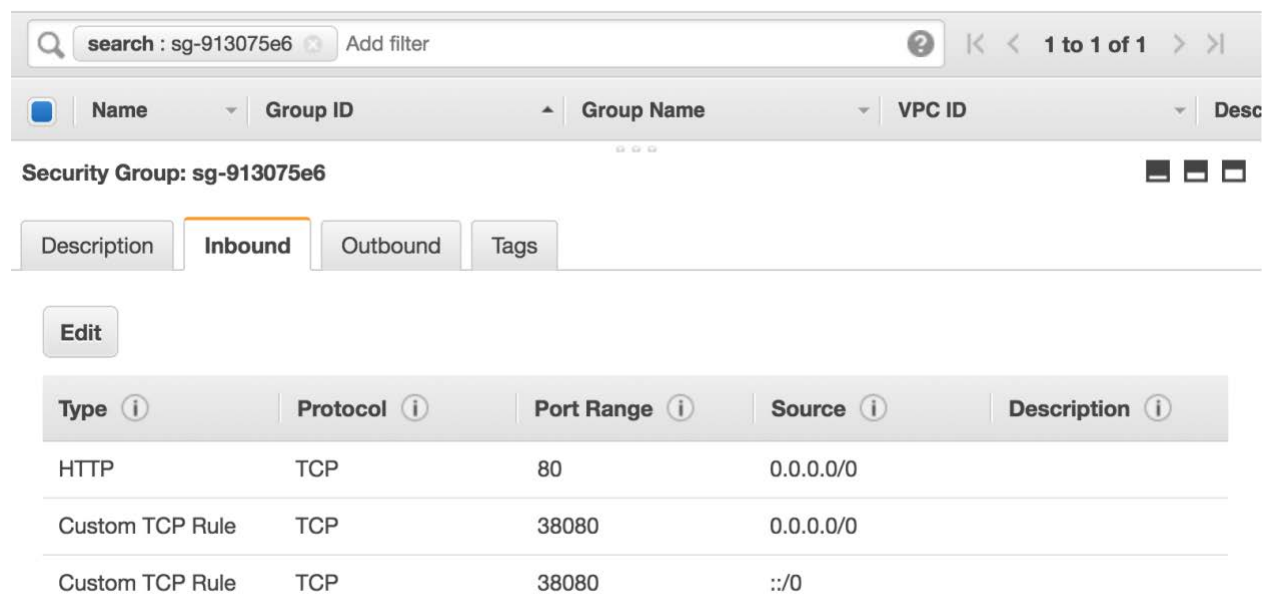
Name	Container Id	Sta	Im:	CP	Ha	Es:
▶ sa4c-emr	1d35fb81-18b3-458b-9cc8-ac0841feb282	0	...	true

Figure 12: AWS - Task Details

For more information about setting up and running tasks, see the topic [“Running Tasks”](#) in Amazon Elastic Container Service Developer Guide (listed in the Recommended Reading section of this paper)

Gotchas – Common Problems When Deploying Containers

My biggest problem with getting containers working are the network settings. Make sure you can access the container from your browser. You might need specific entries in your inbound rules in order to connect from your computer. The port must be opened so that you can connect from your browser. Sometimes specific corporate standards such as firewall settings require further settings in order to make things work. You might need to reach out to your corporate IT folks to see if there is anything additional that you need. Another issue is figuring out where to connect to the running container after you’ve done all the work of deploying your container and you think that it is running. I have depicted a **red circle** around the items showing where to connect to the container and where in your container configuration to find that information.



The screenshot shows the AWS Management Console interface for a Security Group. At the top, there is a search bar with the text "search : sg-913075e6" and "Add filter". Below the search bar, there are tabs for "Description", "Inbound", "Outbound", and "Tags", with "Inbound" being the active tab. An "Edit" button is visible above a table of inbound rules. The table has columns for "Type", "Protocol", "Port Range", "Source", and "Description".

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	
Custom TCP Rule	TCP	38080	0.0.0.0/0	
Custom TCP Rule	TCP	38080	::/0	

Figure 13: AWS - Security Group Settings

Step 6: Connect to SAS Container

Specify the URL that uses the public IP that can be found on the **Details** tab for the cluster definition, and add the port number.

In this example, the URL and port are 54.161.27.146:38080, as shown in Figure 14a and Figure 14b.

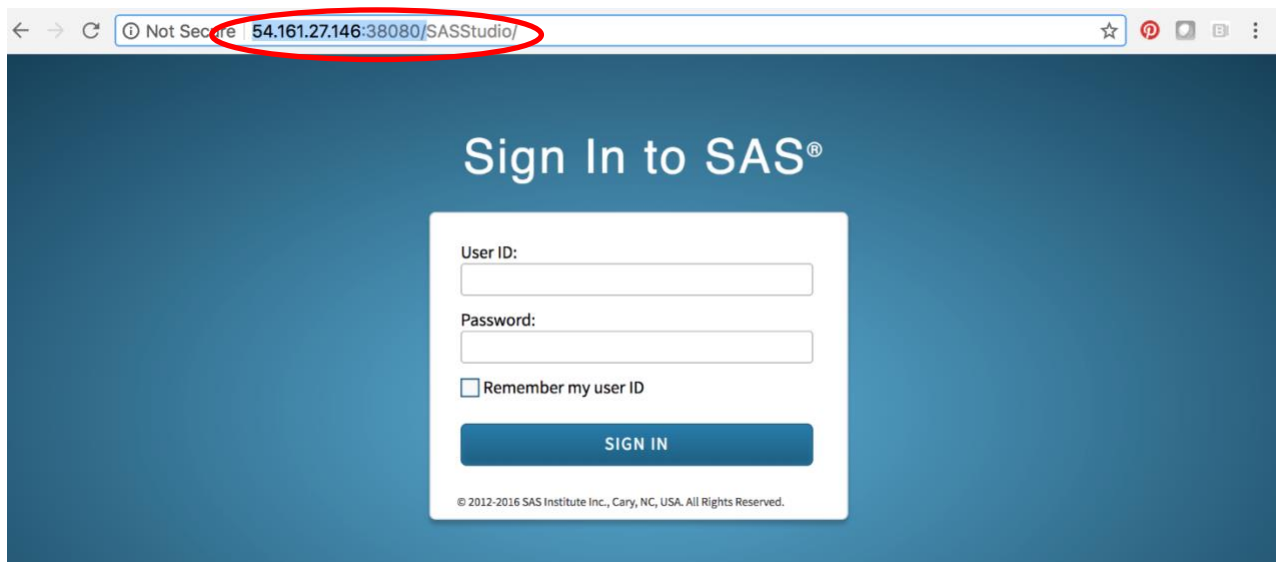


Figure 14a: AWS - Connect to a SAS Container Running in ECS

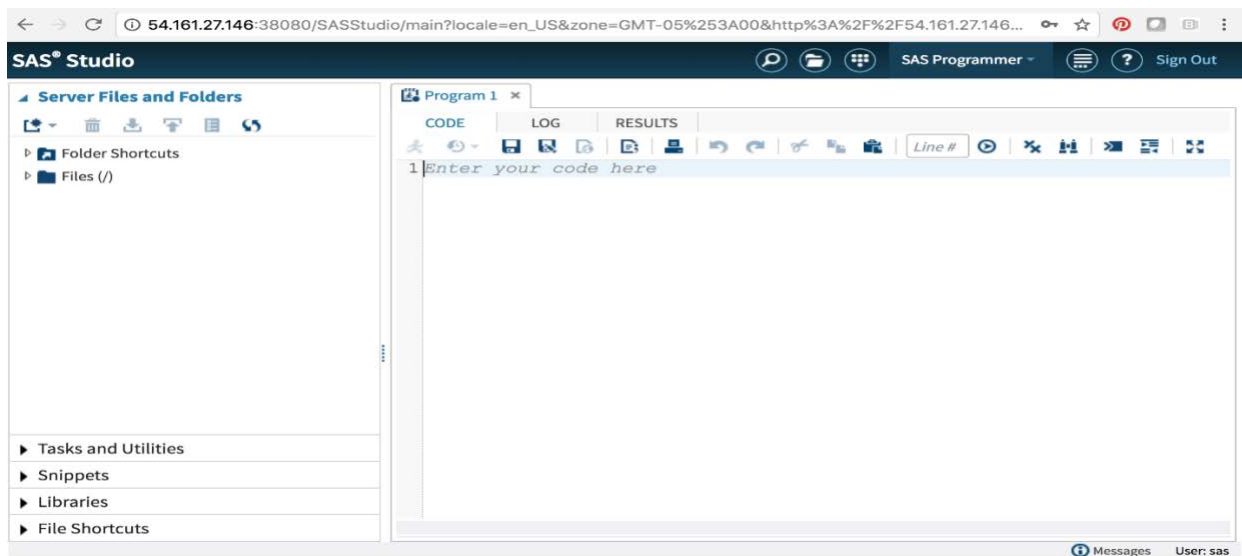


Figure 14b: AWS - Connect to a SAS Container Running in ECS

GOOGLE CLOUD PLATFORM (GCP)

The following two sections on GCP and Azure make it clear that the process of running containers in these cloud environments is very similar. After installing the SDK to talk to the public cloud, you must tag and push your container image to GCP. Figure 15 shows how to verify your gcloud (GCP CLI) environment by using the command `gcloud -version`, which gives all of the details about the Google cloud SDK.

Step 1: Verifying GCP CLI

```
$ gcloud --version
```

Figure 15: GCP - Verifying the CLI

Open a Linux command line where your local Docker repository is accessible. Issue commands to tag the local image to the location in GCP and to push the image to GCP. Figures 16 and 17 show the steps to tag and push to the GCP container repository.

Step 2: Tagging and Pushing Docker Image to GCP Container Repository

```
$ docker tag sa4c:latest us.gcr.io/wwm-wpm/sa4c:latest  
$ gcloud docker -- push us.gcr.io/wwm-wpm/sa4c:latest
```

Figure 16: GCP - Pushing Container to Google Cloud Container Repository

```
!$ gcloud docker -- push us.gcr.io/wwm-wpm/sa4c  
The push refers to a repository [us.gcr.io/wwm-wpm/sa4c]  
7a9eeac0db47: Pushed  
1e8229ec8b88: Pushed  
75a6c523d523: Pushed  
3b1c376ddc97: Pushing [>] 19.74MB/8.001GB  
a6447cff8404: Pushed  
1132d5050a71: Pushed  
72da1c4e1546: Pushed  
70fe22f59807: Pushed  
1a5152f98919: Pushing [=====>] 13.26MB/90.7MB  
d1be66a59bc5: Layer already exists
```

Figure 17: GCP - Command Line Results of Push to Google Cloud Container Registry

Figure 18 shows the Google Cloud Platform dashboard with the image that was pushed from the gcloud CLI.

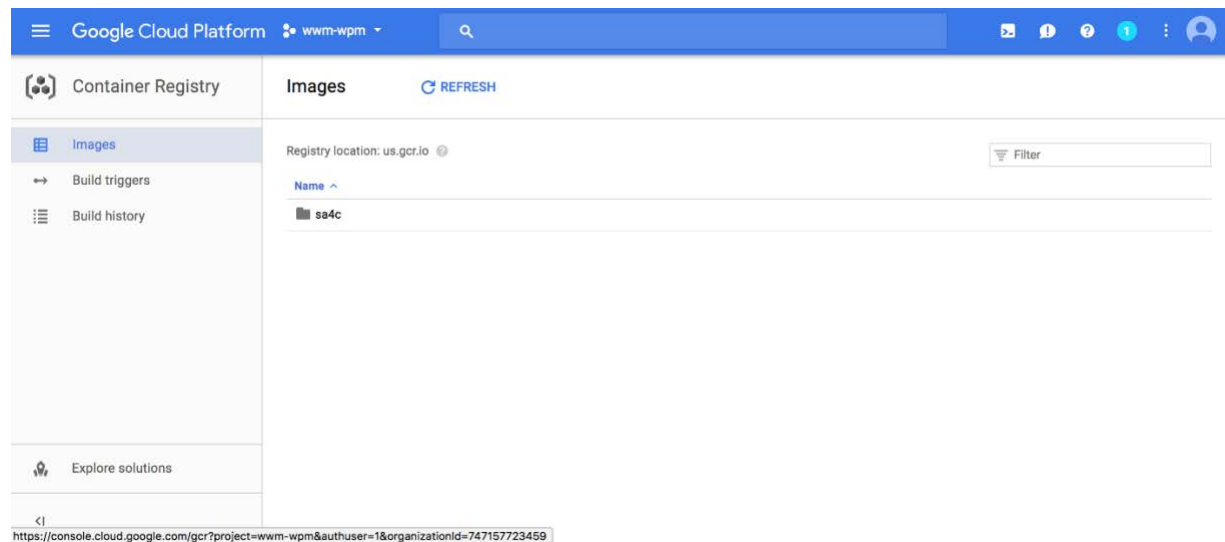


Figure 18: GCP - Dashboard Showing SAS Container in Container Registry

Registering the container in GCP is only the first step. We want to take advantage of the Google-developed Kubernetes orchestrator that deploys, manages, and scales containers. From the GCP

dashboard, create a Kubernetes Engine Cluster. This example is named dd-cluster-1. This cluster has three nodes with eight virtual CPUs (vCPUs) each, and 30 GB memory. It takes a few minutes for the Kubernetes cluster to be prepared. Figure 19 shows the progress of creating the Kubernetes cluster.

Step 3: Create Kubernetes Cluster

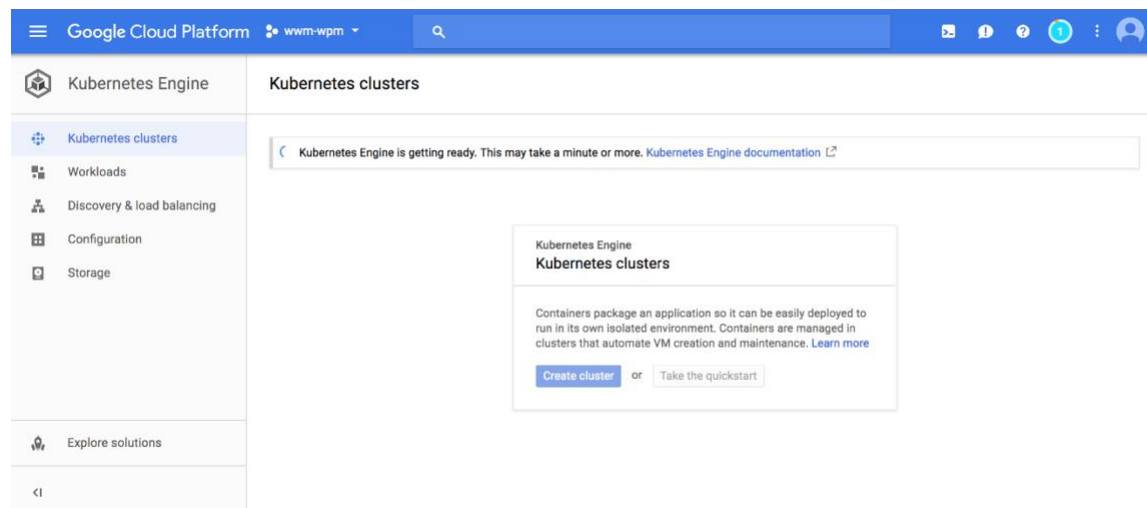


Figure 19: GCP - Kubernetes Cluster to Host Containers

GCP does a nice job of intermingling the dashboard with the command line interface. When we select **Connect** on the dashboard to connect to our cluster, it provides the commands needed to interact with Kubernetes. It essentially provides an endpoint between the Linux command line environment and the GCP Kubernetes environment with one command. The command in Figure 20 was provided by the dashboard, and it can then be plugged into the local command line environment.

```
$ gcloud container clusters get-credentials dd-cluster-1 --zone us-east1-b --project www-wpm
```

Figure 20: GCP - Kubernetes Console and CLI Integration

Step 4: Connect to Kubernetes

You can also set the environment to have defaults to reduce the amount of typing required. Figure 21 shows the compute/zone being set to default to us-east1-b. Figure 22 issues the endpoint command and then starts the SAS container to run on dd-cluster-1.

Step 5: Run SAS Container and Get Address of Running Container

Figure 21 and 22 shows the commands to run the SAS container. Figure 23 shows the command to get the public IP address of the running container.

```
donna_decapite@www-wpm:~$ gcloud config set compute/zone us-east1-b
Updated property [compute/zone].
donna_decapite@www-wpm:~$ gcloud container clusters get-credentials dd-cluster-1
Fetching cluster endpoint and auth data.
kubeconfig entry generated for dd-cluster-1.
donna_decapite@www-wpm:~$ kubectl run sas-server --image=us.gcr.io/www-wpm/sa4c:latest --port 38080
deployment "sas-server" created
donna_decapite@www-wpm:~$
```

Figure 21: GCP - Kubernetes Run SAS Container


```
donna_decapite@wvm-wpm:~$ kubectl expose deployment sas-server --type="LoadBalancer"
service "sas-server" exposed
donna_decapite@wvm-wpm:~$
```

Figure 22: GCP - Create a Service Object That Exposes the SAS Container Deployment

```
donna_decapite@wvm-wpm:~$ kubectl get service sas-server
NAME         TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
sas-server   LoadBalancer 10.19.245.182 104.196.172.199 38080:30952/TCP 1m
donna_decapite@wvm-wpm:~$
```

Figure 23: GCP - Get Detailed Information about the SAS Kubernetes Service – Public IP Address

Step 6: Connect to SAS Container

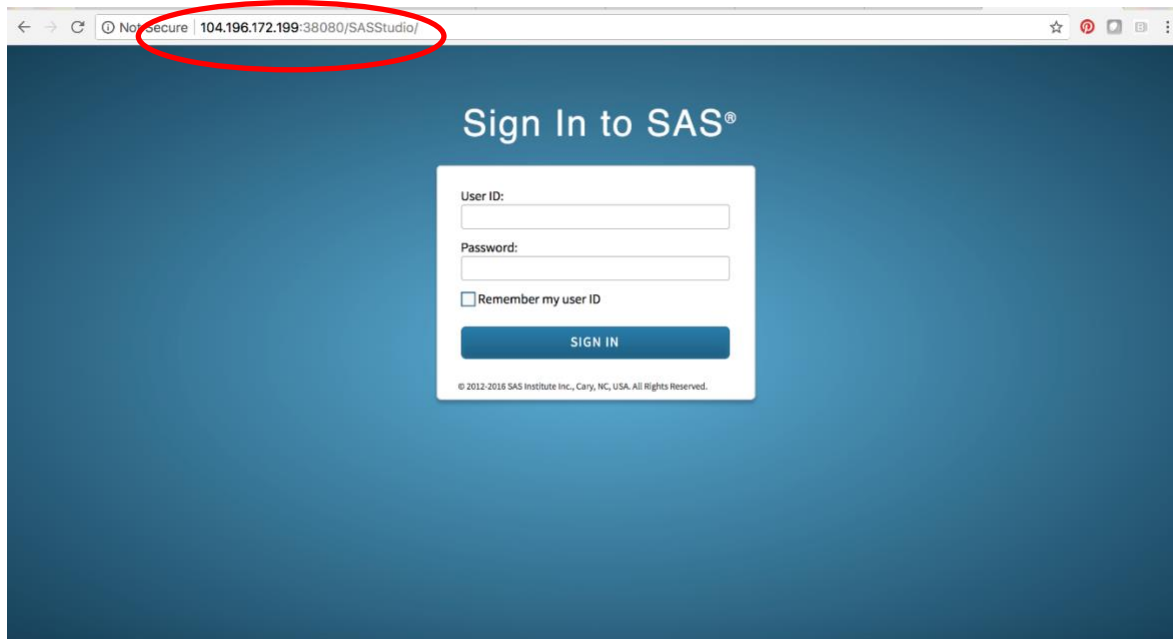


Figure 24: GCP – Connect to SAS Container

MICROSOFT AZURE

The prerequisite step for all the public cloud providers is to get a CLI installed. I had a bit of trouble with one of the prerequisite steps for the Azure CLI when trying to install xcode-select.

From the Macintosh command line, I used the command `xcode-select -install`.

```
$ xcode-select -install
```

Figure 25: Azure – Prerequisite Step to Install CLI

I had problems getting Xcode from the command line and found going to Apple's App Store and downloading Xcode was my best option. This step took quite some time, so it is best to start this download before a meeting or lunch break. After accepting the Xcode license, I was able to perform a "brew upgrade". After that I was able to continue with the process of installing the Azure CLI.

After you install the Azure CLI, use the commands in Figure 26 to validate your environment.

```
$ az --version
$ az login
```

Figure 26: Azure – Verify the CLI and Log into Azure

When you enter the az login command, you are directed to enter a URL with an access code to validate the login information.

At this point you are authenticated and can work within your Azure project associated with your account.

MAKING THE CASE FOR PERSISTENT STORAGE

Note: All the cloud providers support persistent storage. I chose to demonstrate that integration with Azure, but could have shown that capability with any of the providers.

For Azure, this example assigns persistent storage that resides outside of the container. This storage can be used while the container is running, but most importantly, anything that is written there does not go away when the container stops running. The idea is that you want to ensure that data is not lost, such as files that you want to work with while you're running SAS and output that you want to keep. Containers need to have a definition for persistent storage to make this happen. In the world of the cloud, resources are shut down when they are no longer needed and can be quickly restarted when the demand arises. If we do not associate persistent storage with the container deployment, then all the data that was written during the container session is lost.

For this environment, the environment variables are set to make the commands easier to manage.

```
ACI_PERS_RESOURCE_GROUP=d-resource-group
ACI_PERS_STORAGE_ACCOUNT_NAME=mystorageaccount$RANDOM
ACI_PERS_LOCATION=eastus
```

Figure 27: Azure – Setup Environment to Interact with Azure

This Azure example includes external storage configured for the container.

Note: All of the cloud providers support external storage, and each has a different approach to defining the storage that can be found on the cloud providers website.

The following figures list the az commands used to allocate and use this storage.

First, create the storage account, as shown in Figure 28.

```
$ az storage account create \
  --resource-group $ACI_PERS_RESOURCE_GROUP \
  --name $ACI_PERS_STORAGE_ACCOUNT_NAME \
```

Figure 28: Azure – Create Storage Account

Next, export the connection string as an environment variable. The command az storage share create references this environment variable when creating the Azure file share. See Figure 29.

```
$ export AZURE_STORAGE_CONNECTION_STRING=`az storage account show-connection-string -
-resource-group $ACI_PERS_RESOURCE_GROUP --name
$ACI_PERS_STORAGE_ACCOUNT_NAME --output tsv`
```

Figure 29: Azure – Export Storage Connection String

Create the file share, as shown in Figure 30.

```
$ create -n $ACI_PERS_SHARE_NAME
$ az storage share create -n $ACI_PERS_SHARE_NAME
```

Figure 30: Azure – Create File Share

Set the Storage account and key value. See Figure 31 and Figure 32.

```
$ STORAGE_ACCOUNT=$(az storage account list --resource-group
$ACI_PERS_RESOURCE_GROUP --query
"[?contains(name,'$ACI_PERS_STORAGE_ACCOUNT_NAME')].[name]" --output tsv
$ echo $STORAGE_ACCOUNT
```

Figure 31: Azure – Setup Storage Account

```
$ STORAGE_KEY=$(az storage account keys list --resource-group
$ACI_PERS_RESOURCE_GROUP --account-name $STORAGE_ACCOUNT --query "[0].value" --
output tsv)
$ echo $STORAGE_KEY
```

Figure 32: Azure – Setup Storage Key

Figure 33 shows the command to create a container with the appropriate volume and mount settings.

```
$ az container create \
  --resource-group $ACI_PERS_RESOURCE_GROUP \
  --name sas \
  --image dodeca.azurecr.io/sa4c-dd \
  --ip-address Public \
  --ports 38080 \
  --cpu 2 \
  --memory 8 \
  --azure-file-volume-account-name $ACI_PERS_STORAGE_ACCOUNT_NAME \
  --azure-file-volume-account-key $STORAGE_KEY \
  --azure-file-volume-share-name $ACI_PERS_SHARE_NAME \
  --azure-file-volume-mount-path /dodeca/mysas/
```

Figure 33: Azure – Create SAS Container with External Storage Outside the Container

Notice the file volume mount is used. This location is the container for SAS files and data sets.

SIDEBAR: OTHER EXAMPLES OF USING MOUNT POINTS

In another example, in my Dockerfile I made a directory to point to my Hadoop JAR files and then added the JAR files to that directory. In this case, I added a shell script that defined predefined LIBNAME statements, which I then called in my entrypoint shell script. I could have used a similar technique to “load” things such as SAS formats, samples, or my Home directory with SAS data sets and files. Mounts are also used to point to LDAP servers to provide container authentication. The following snippet shows how I mounted the Hadoop Client Jars into my container needed to run **SAS/ACCESS Interface to Hadoop**:

```
RUN mkdir -p /usr/local/SASHome/hadoopjars
ADD emrjars.gz /usr/local/SASHome/hadoopjars
ADD emrsite.gz /usr/local/SASHome/hadoopjars
ADD addins.sh /usr/local/SASHome
```

SHOW PROPERTIES OF AZURE CONTAINER

Figure 34 shows the command used to show the properties of the container. Azure prompts for the image registry password, which is in the Access keys container registry.

```
$ az container show --resource-group $ACI_PERS_RESOURCE_GROUP --name sas --output table
```

Figure 34: Azure – Show Properties of SAS Container

Figure 35 shows the output of this command. Notice the public IP address and ports. This is the address needed to connect to the container.

```
$ az container show --resource-group $ACI_PERS_RESOURCE_GROUP --name sas --output table
Name      ResourceGroup  ProvisioningState  Image                                     IP:ports      CPU/Memory      OsType  Location
-----
sas       d-resource-group  Succeeded          dodeca.azurecr.io/sa4c-dd              52.170.195.42:38080  2.0 core/8.0 gb  Linux  eastus
```

Figure 35: Azure – Output of Properties Command

Figure 36 shows what it looks like when a container is provisioned and ready to use. It includes the address and IP:Ports used to connect to the running container instance.

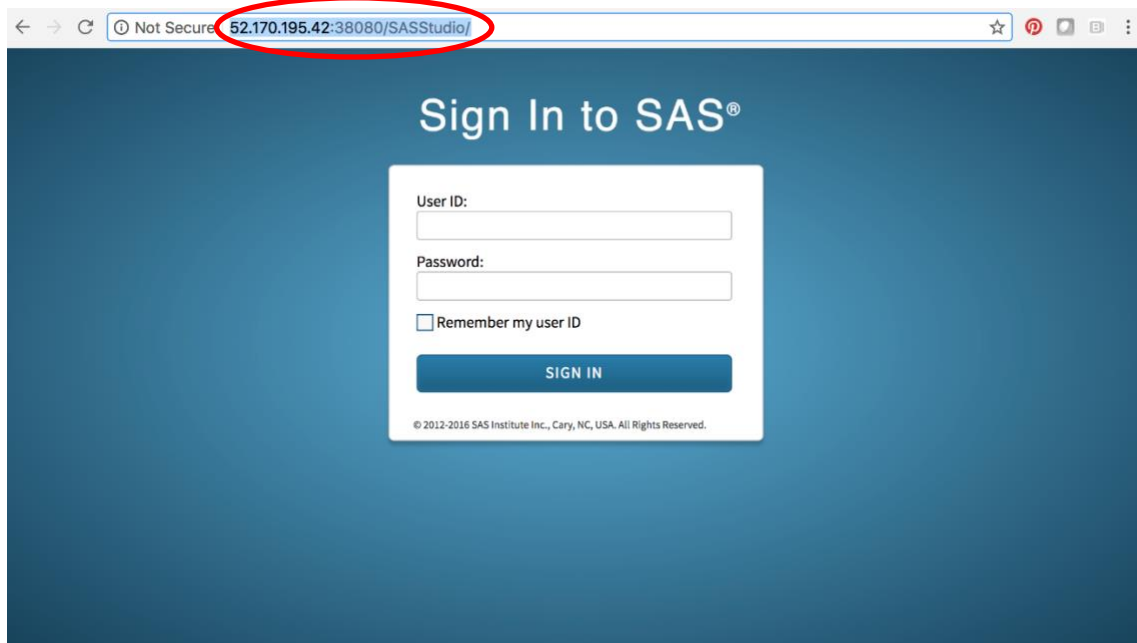


Figure 36: Azure – Connect to SAS Container

CO-MINGLING CONTAINERS – SAS VIYA, JUPYTER NOTEBOOK, PYTHON, R

Many customers are putting more than one application into their containers. Figure 37 shows a snippet of the Dockerfile that illustrates how you can comingle Anaconda, SASPy, SAS Scripting Wrapper for Analytics Transfer (SWAT), R Studio and seaborn library. These additional components were added to my SAS Viya container. The idea is to create one large container that can do all these things. It essentially provides a data scientist toolkit of various capabilities such as these.

- SASPy – SAS Python Integration (<https://github.com/sassoftware/saspy>)
- SWAT - The SWAT package is the Python client to SAS Viya Cloud Analytic Services (CAS). It allows users to execute CAS actions and process the results all from Python.
- R
- Python
- Additional libraries (seaborn)

```

#####
# Install Anaconda & R
#####
# Install Jupyter prerequisites
RUN yum -y install bzip2 wget git libpng12-devel urw-fonts \
    && yum clean all

# Set the install location & add to the PATH
ENV INSTALL_DIR /opt/anaconda
ENV PATH $PATH:$INSTALL_DIR/bin

# Download and install Anaconda
RUN wget https://3230d63b5fc54e62148e-c95ac804525aac4b6dba79b00b39d1d3.ssl.cf1.rackcdn.com/Anaconda3-2.5.0-
Linux-x86_64.sh \
    && chmod 755 Anaconda3-2.5.0-Linux-x86_64.sh \
    && ./Anaconda3-2.5.0-Linux-x86_64.sh -b -p $INSTALL_DIR \
    && rm -f Anaconda3-2.5.0-Linux-x86_64.sh

# Install additional R and Python packages
RUN yum install -y R libcurl-devel cairo-devel \
    && yum clean all
#####
# Install SAS Kernel & SASPy
#####
# Install the packages
RUN $INSTALL_DIR/bin/pip install saspy sas_kernel pipefitter
RUN sed -i 's|/opt/sasinside/SASHome/SASFoundation/9.4/bin/sas_u8|/opt/sas/viya/home/SASFoundation/bin/sas_u8|g'
/opt/anaconda/lib/python3.5/site-packages/saspy/sascfg.py
#####
# Install SWAT
#####
# Install from public GitHub page
# NOTE: Install into the Anaconda env not the default RHEL env
RUN $INSTALL_DIR/bin/pip install https://github.com/sassoftware/python-swat/releases/download/v1.2.1/python-swat-1.2.1-
linux64.tar.gz
RUN echo "install.packages(c('http', 'jsonlite', 'repr', 'IRdisplay', 'evaluate', 'crayon', 'pbdZMQ', 'devtools', 'uuid',
'digest', 'ggplot2', 'formatR', 'highr', 'markdown', 'yaml', 'htmltools', 'caTools', 'bitops', 'knitr', 'base64enc', 'projroot', 'rmarkdown'),
repos='http://cran.us.r-project.org'); devtools::install_github('IRkernel/IRkernel')" > /tmp/setup.R \
    && R CMD BATCH /tmp/setup.R /tmp/setup.out \
    && cat /tmp/setup.out \
        && wget -O /tmp/r-swat-1.0.0-linux64.tar.gz https://github.com/sassoftware/R-swat/releases/download/v1.0.0/r-swat-
1.0.0-linux64.tar.gz \
    && R CMD INSTALL /tmp/r-swat-1.0.0-linux64.tar.gz
#####
# Install RStudio
#####
RUN wget https://download2.rstudio.org/rstudio-server-rhel-1.0.153-x86_64.rpm \
    && yum install -y --nogpgcheck rstudio-server-rhel-1.0.153-x86_64.rpm \
    && rm -f rstudio-server-rhel-1.0.153-x86_64.rpm
#####
# install seaborn
#####
# Install additional Python packages
RUN conda install -y seaborn

# Expose only port 80
# SAS Studio & Jupyter are proxied behind port 80
EXPOSE 80

```

Figure 37. snippet to add additional capabilities to Docker container

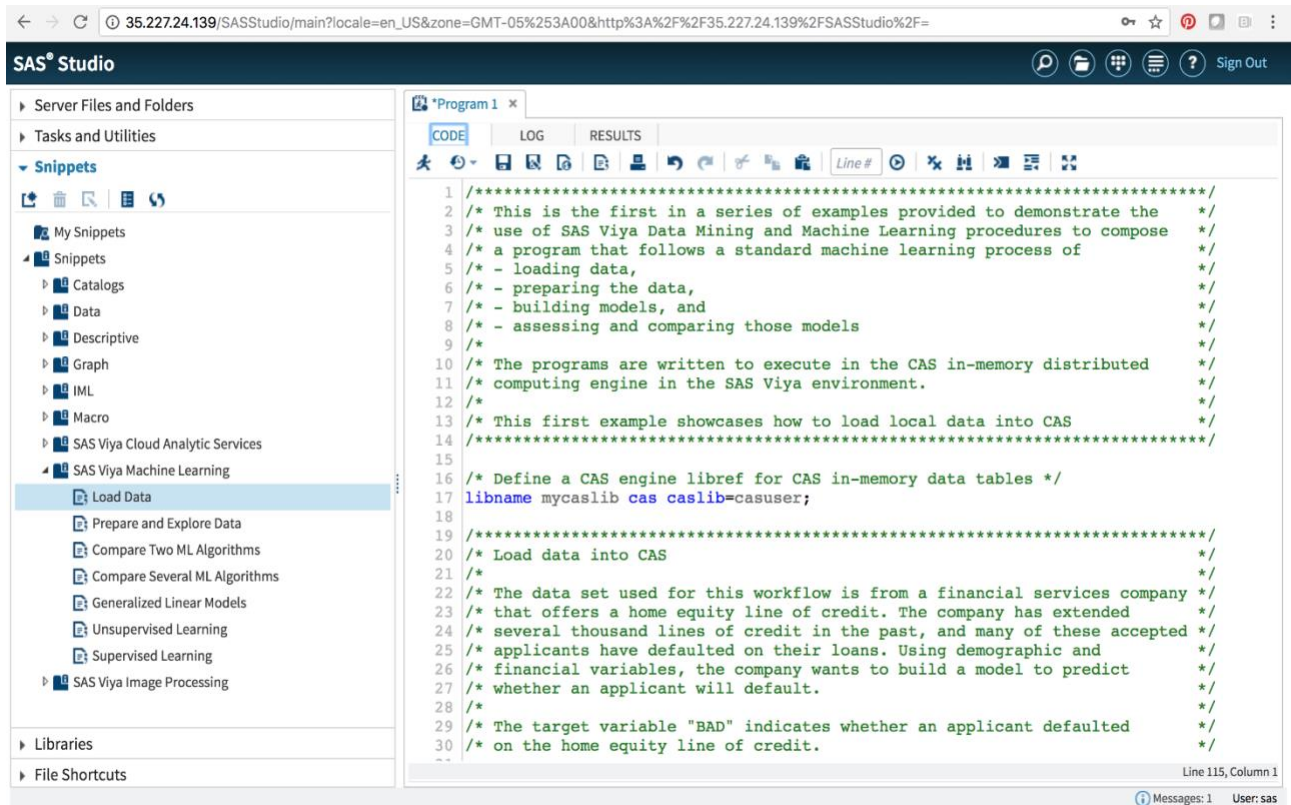


Figure 38: SAS Studio Leveraging CAS Capabilities



Figure 39: Leveraging Jupyter Notebook from the Container

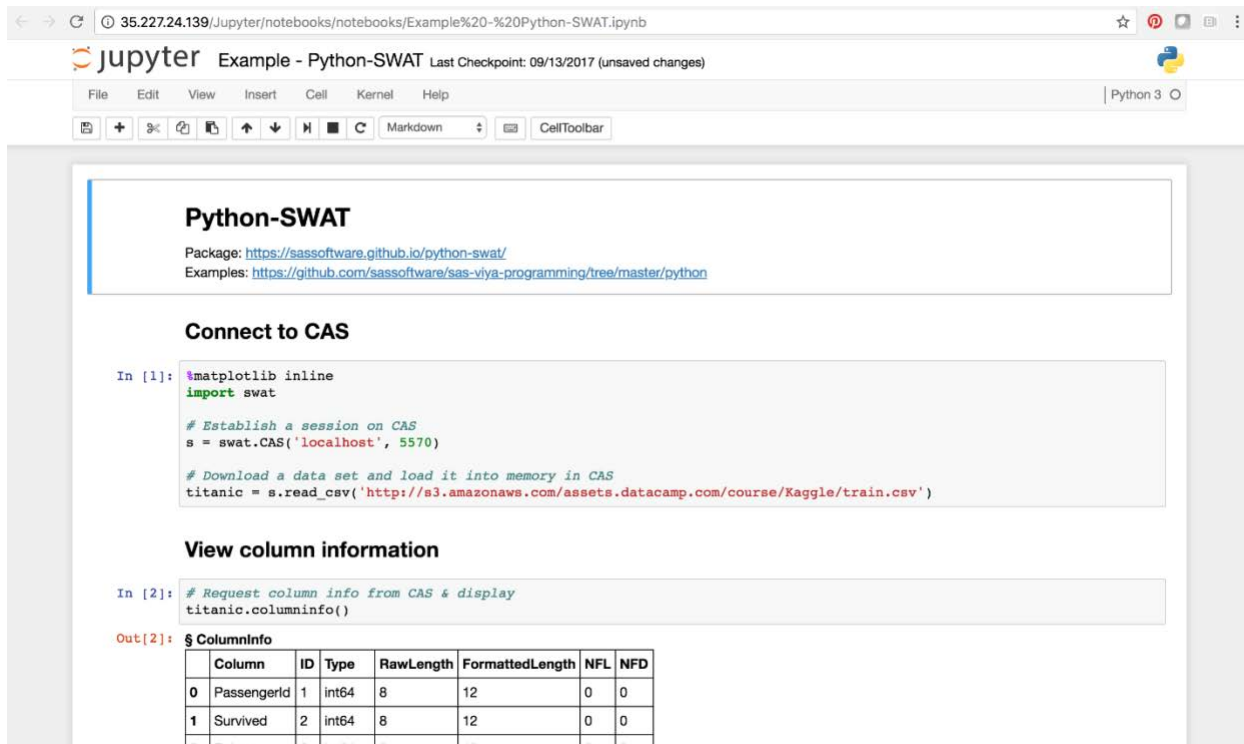


Figure 40: Leveraging Python SWAT Notebook from the Container

AWS – EMR INTEGRATION

The next example is accessing EMR Hadoop data from the container running on AWS. Follow the steps outlined earlier for pushing and running the container. Define the AWS ECS environment to use a Virtual Private Cloud and install the AWS EMR in the Virtual Private Cloud. You can use the quick start path in order to configure AWS EMR. By using the Dockerfile techniques mentioned earlier in this document that injected the Hadoop JAR files into the container, you can point to the Hadoop JAR file path and issue a LIBNAME statement to EMR from your SAS Client (SAS Studio). The figures below show some simple steps to connect to EMR Hadoop and create a table.

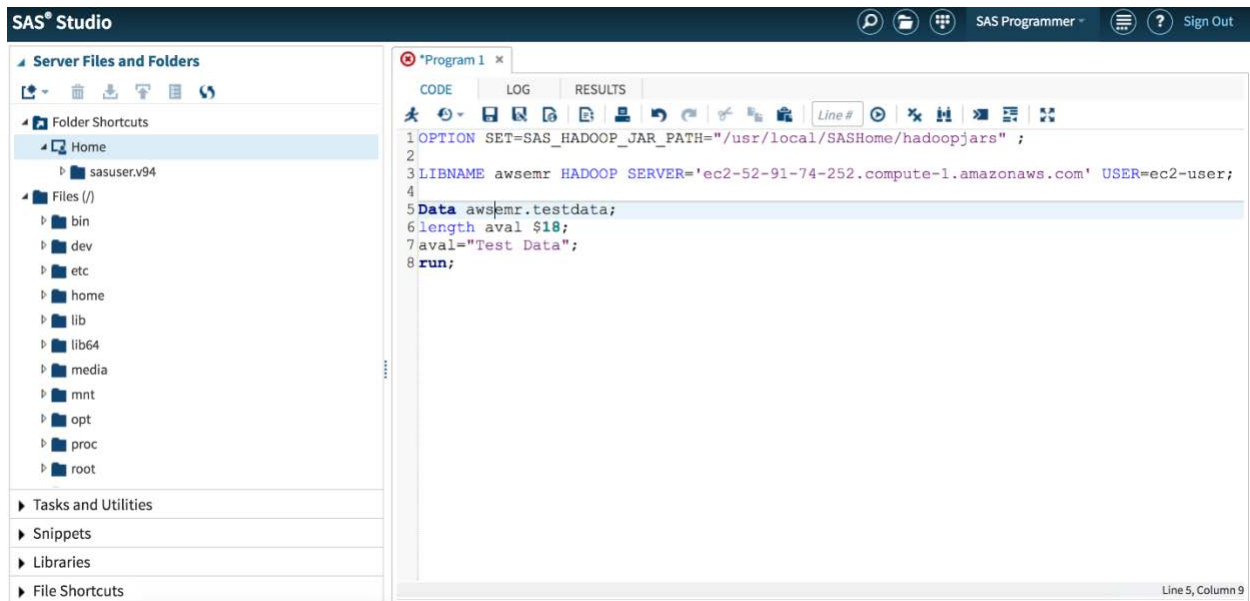


Figure 41: Leveraging EMR from SAS Container Running in AWS

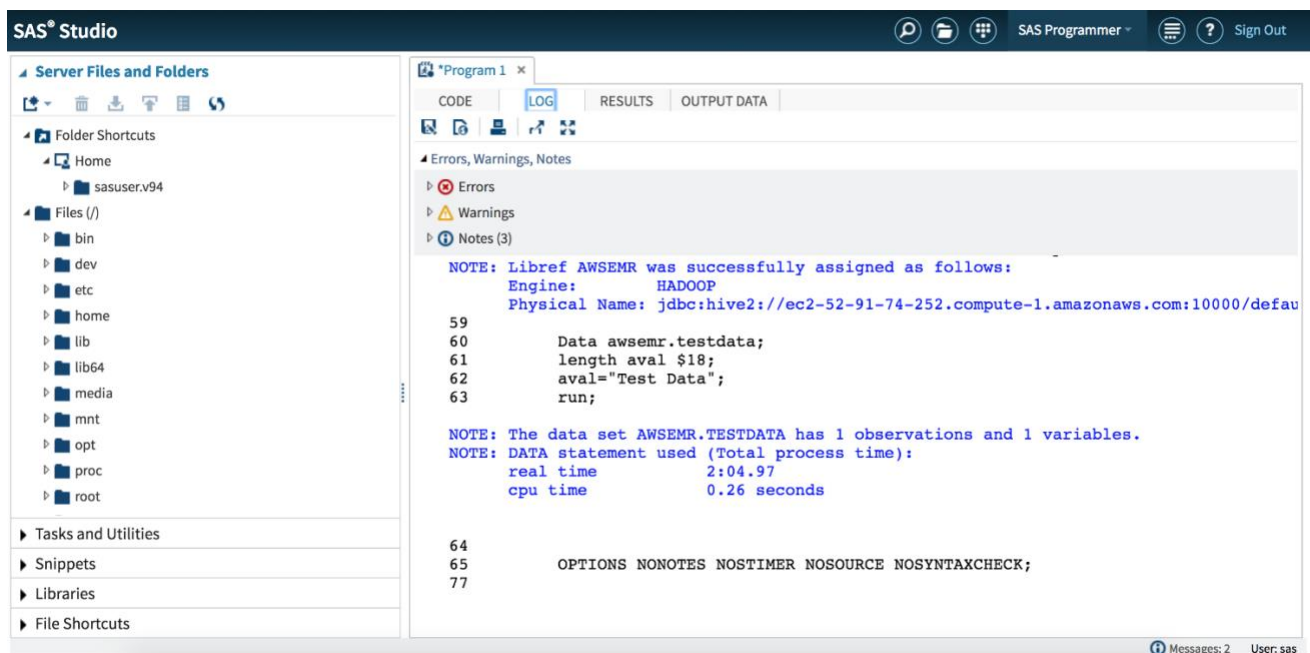


Figure 42: Leveraging EMR from SAS Container Running in AWS

OPENSIFT

Red Hat Openshift provides another container environment that can be leveraged by customers. The big difference between Openshift and the other public cloud providers is that OpenShift runs on-premise in a private cloud, but it can also leverage hybrid cloud environments (both public and private cloud). OpenShift uses Kubernetes as its container orchestrator.

OpenShift has a CLI like all the other vendors, but OpenShift also has a UI that allows you to easily create projects and containers and perform most other tasks. The only requirement for the CLI is to override some of the project settings. The pattern for deploying containers is very simple.

1. Create a project.

2. Add a container to the project.
3. Define the characteristics of how to run the container.

Figures 43 – 46 show these steps.

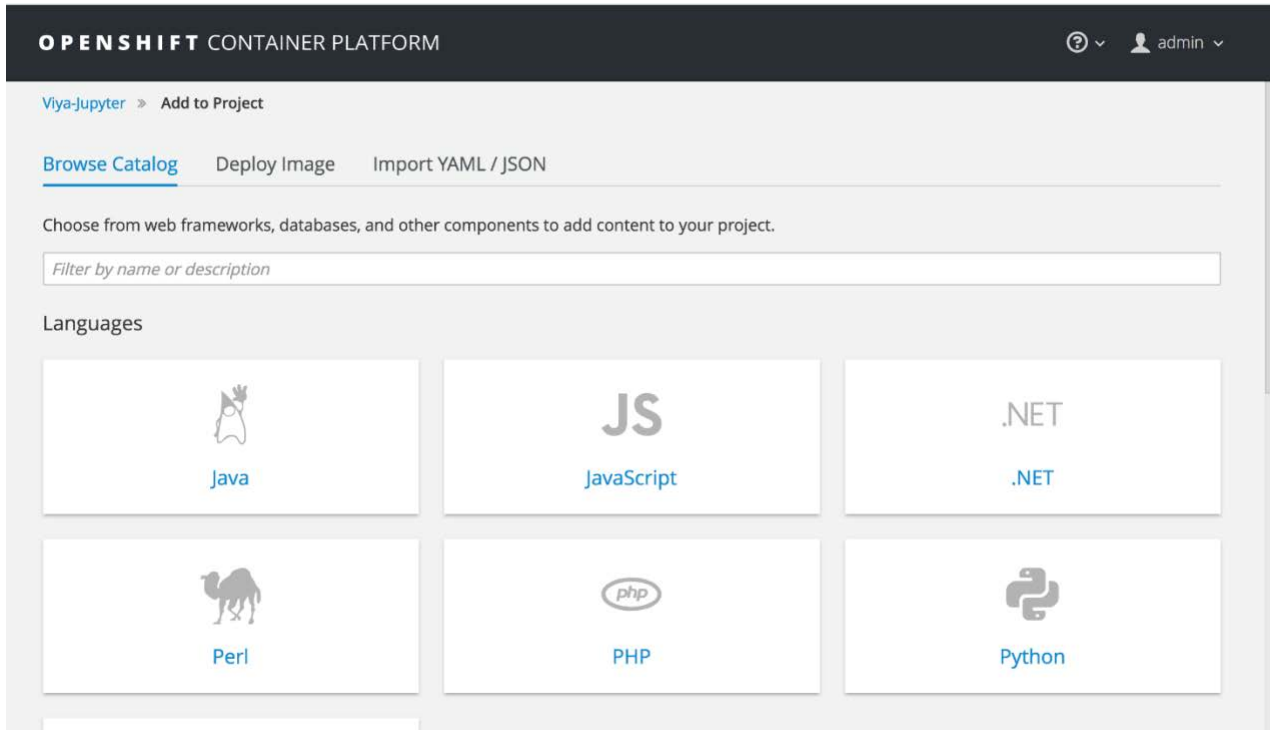


Figure 43: Adding to an Openshift Project

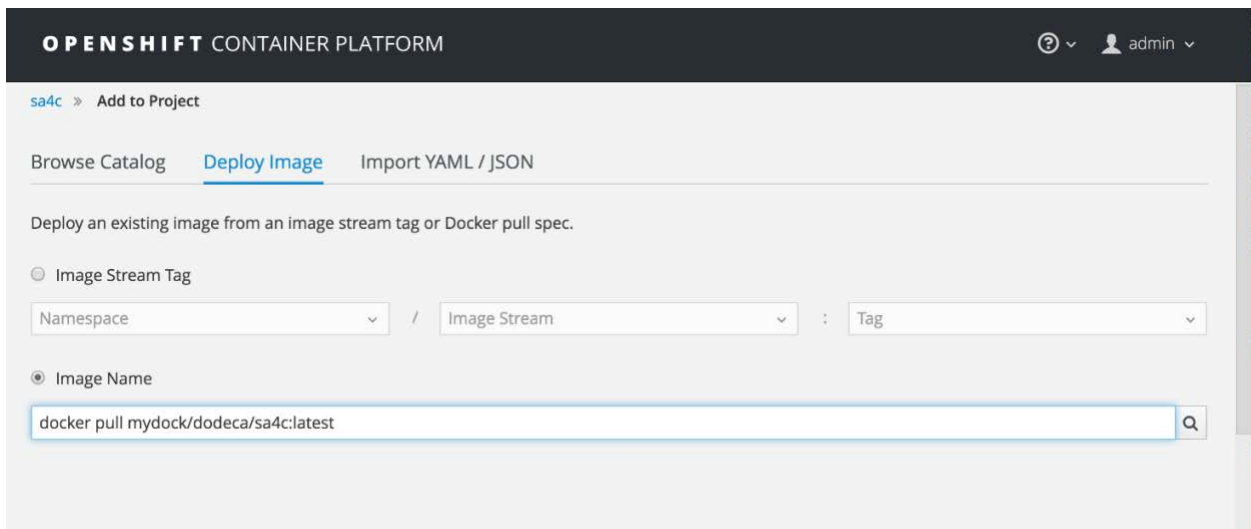


Figure 44: Adding Docker Image to an Openshift Project

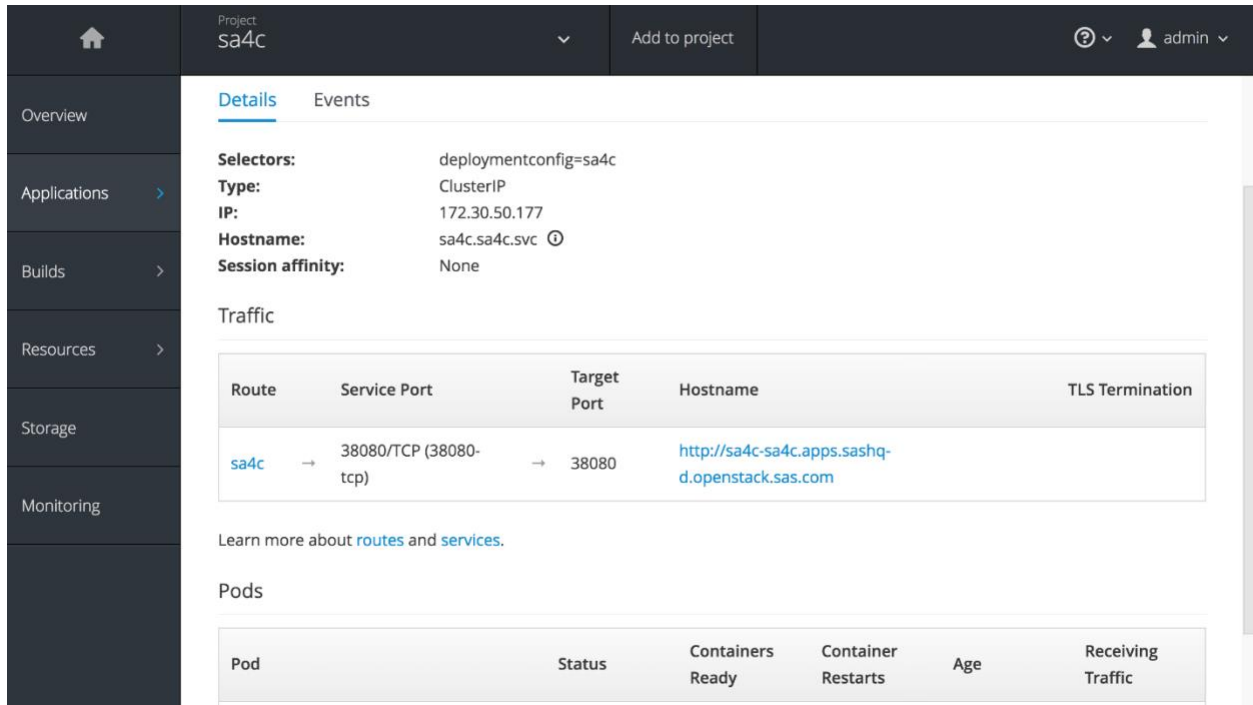


Figure 45: Defining Characteristics of Container – Port Forwarding

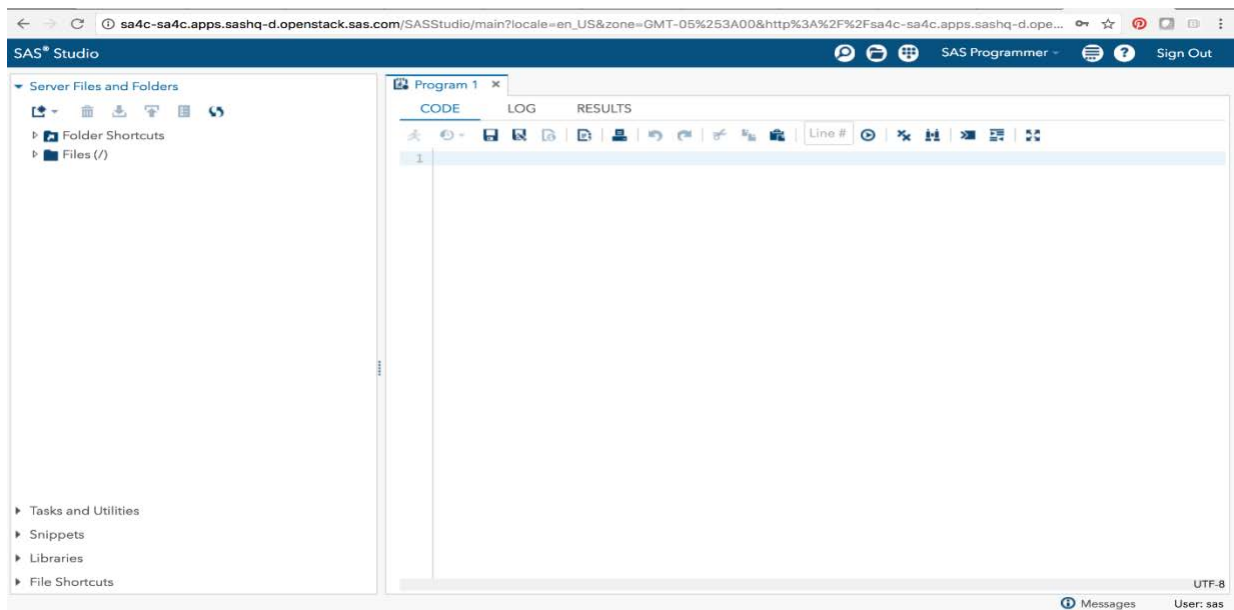


Figure 46: Launch SAS Studio from a Running OpenShift Container

CONCLUSION

The goal of this paper was to show the steps needed to run containers in the cloud. After you have completed the work of getting things registered, it is very easy to quickly leverage cloud environments. This paper contains information gathered from working with building containers, deploying them to various cloud infrastructures, and working with Hadoop. Given the fast-moving technology of containers, this paper might be updated in the future. For notifications of these updates along with other container tuning guidelines, please subscribe to the SAS Administration and Deployment Community (go to https://communities.sas.com/t5/Administration-and-Deployment/bd-p/sas_admin).

ACKNOWLEDGMENTS

Many thanks to Douglas Liming, Jonah Justice, and Jonathan Walker for helping with this paper.

RECOMMENDED READING

Orchestrators

- "Smooth Sailing with Kubernetes." Google. Available <https://cloud.google.com/kubernetes-engine/kubernetes-comic/>.
- Burt, Paul. "What is Kubernetes? An Intro for Beginners." CoreOS. Available <https://coreos.com/blog/what-is-kubernetes.html>
- Amazon Web Services. 2018. "Running Tasks." In *Amazon Elastic Container Service Developer Guide*. Amazon Web Services Inc. Available https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs_run_task.html.
- De Capite, Donna. 2017. "Convergence of Big Data, The Cloud, and Analytics: A Docker Toolbox for the Data Scientist." Proceedings of the SAS Global Forum 2017 Conference. Cary, NC: SAS Institute Inc. Available :<http://support.sas.com/resources/papers/proceedings17/SAS0687-2017.pdf>

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Donna De Capite
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.

Donna.DeCapite@sas.com

<http://www.sas.com>

@geekyDonna

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

Disclaimer of Warranty

Any samples provided are "as is" without any warranties, express or implied, including but not limited to implied warranties or merchantability and/or fitness for a particular purpose. The Institute and its licensor(s) disclaim any liability connected with the use of the instruction. The Institute offers no technical support for the instruction.

Limitation of Liability

The Institute and its licensor(s) are not liable for (a) incidental, consequential, special, or direct damages of any sort, whether arising in tort, contract or otherwise, even if the Institute has been informed of the possibility of such damages, or (b) any claim by any other party. Some jurisdictions do not allow the exclusion or limitation of liability for incidental or consequential damages, so this limitation and exclusion might not apply to you.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.