OpenFOAM in AWS

Creating a cluster

Grzegorz Kotysz

January 25, 2019

Contents

1 Introduction

The purpose of this paper is to learn how to use AWS computing cloud to run OpenFOAM cases using CLI, setting up cloud computing cluster and transferring results to the local computer. Simulation case will be based on dam break tutorial case.

2 CFD Direct AWS Tutorial[1]

2.1 Instance setup[2] and launch[3]

Before any interaction with AWS instances, user has to create an account and then create Key Pair, which will be used to authorize connection with the instance. It is recommended that it is saved in .ssh directory. Appropriate permissions also need to be set, so that only user can read this file. It can be done via terminal by executing following command:

```
chmod 400 path/file
```

Launching an Instance will be based on the one prepared by CFD Direct with OpenFOAM preinstalled. It can be accessed via `https://aws.amazon.com/marketplace/pp/B017AHY016/`. Then click "Continue to Subscription" to be able to launch Instances. Next click "Continue to Configuration". Here user can choose version of software and select region. Next click "Continue to Launch". Here user specifies whether to launch the Instant through EC2 or from Website. In our case it is EC2 Console. Now the preferred instance is chosen, for the sake of tutorial free t2.micro is used. Click "Review and Launch". In the new window find "Edit Security Groups" and in "Source" drop-down menu select "My IP" option. Click "Review and Launch". Similarly one can edit storage size by selecting "Edit Storage", setting desirable storage and clicking "Review and Launch". The next step is to launch the instant by clicking "Launch" button, select appropriate Key Pair and click "Launch Instances". Instance should be initialized and user should note IPv4 Public IP address.

2.2 Connecting to an Instance and testing OpenFOAM[4]

Connection with the instance can be established via SSH by running following command in the terminal:

```
ssh -i path/SSHkey instanceUsername@publicIP
```

Note: default username is "ubuntu". User is presented with terminal prompt. Now OpenFOAM can be tested. It will be done by changing to $FOAM_RUN, copying *pitzDaily* case from *tutorials* directory, changing to copied case directory, generating mesh and running *simpleFoam* solver. This can be done by following commands:

```
run
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily .
cd pitzDaily
blockMesh
simpleFoam
```

Finally, data can be transferred between local machine and remote instance by scp command, i.e. copying from instance to local machine:

```
scp -i path/SSHkey instanceUsername@publicIP:path/file localPath
```

and from local machine to remote instance:

```
scp -i path/SSHkey localPath/file instanceUsername@publicIP:path/
```

[1] CFD Direct, URL: `https://cfd.direct/cloud/aws/`, Accessed: 25.10.2018
[2] CFD Direct, URL: `https://cfd.direct/cloud/aws/setup`, Accessed: 25.10.2018
[3] CFD Direct, URL: `https://cfd.direct/cloud/aws/launch`, Accessed: 25.10.2018
[4] CFD Direct, URL: `https://cfd.direct/cloud/aws/connect`, Accessed: 25.10.2018

2.3 Creating a CFD Cluster[5]

The example presented in the tutorial uses c4.8xlarge instances, however t4.micro can be used for testing purpose if no placement group is defined. If placement group is used, one must check whether chosen instance type can be used as a part of the placement group[6]. Cluster will be based on 1 master instance with appropriate storage for the simulation and 3 slave instances with minimal storage.

Note: Placement Group determines how instances are placed within Amazon hardware. For best performance, instances should be placed within *Cluster* group, which ensures that instances are places in low-latency group.[7]

The one time configuration of security group is needed to enable access between instances. At this point it will be created from GUI.

1. Select *Security Groups* in AWS console.
2. Click *Create Security Group*.
3. In the pop-up windows specify *Group Name* and *Description* which will specify the role for the group.
4. Choose *Add Rule* option and for *Inbound* select **SSH** Type and **My IP** Source.
5. Click *Create*.

In order to allow instances to connect with each other within cluster, one must edit created security group.

1. Select desired security group and for *Inbound* rules tab click *Edit*.
2. Click *Add Rule* and select *All TCP* Type.
3. Select **Custom** Source and start typing security group name until a panel pops up. Then select your *cluster group*.

To ensure smooth and fast operation of a cluster, instances need to be connected through a low latency, high speed network. It can be achieved by running them in appropriate placement group.

1. In AWS console select *Placement Groups*.
2. Choose *Create Placement Group*.
3. Enter group name, select **cluster** in *Strategy* field and click *Create*.

The next step is to launch instances. In the tutorial instances are created one at a time, but placement group documentation suggests to create all the cluster's instances at once to ensure that there is enough capacity on a given server. The procedure is as follows:

1. Choose an instance that supports placement groups.
2. Under *Instance Details* select *group_name* from the *Placement Group* menu.
3. Under *Security Group* choose *Select and existing security group* and choose appropriate group.
4. Under *Storage* set a volume size which will be enough to store the data.
5. Click *Launch*.

Slave instances can be created all at once by specifying number of instances under *Instance Details*. For the slave instances disable *Auto-assign Public IP* under *Instance Details* as these instances only need to communicate with the master.

[5] CFD Direct, url: `https://cfd.direct/cloud/aws/cluster`, Accessed: 25.10.2018

[6] AWS, URL: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html`, Accessed: 08.11.2018

[7] Amazon, URL: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html`, Accessed: 26.12.2018

2.3.1 Connecting instances

Once per login session user must enable SSH access from master instance to the slave instance by using *SSH agent forwarding*. It allows using to use agent forwarding instead of storing key on master node, which is not advised. Agent forwarding is enabled by adding private key to the authentication agent by the following command:

```
ssh-add path/SSHkey
```

Now user can SSH login without specifying key by using -A option:

```
ssh -A ubuntu@master_ip
```

Cluster is set up in a way that all data is stored on a volume attached to the master instance. Hence, OpenFOAM directory on master instance must be shared across slave instances using network file system (NFS) protocol. Firstly OpenFOAM directory is exported with *exportfs* (one time only) and then mounted by all slave instances using *mount*. OpenFOAM directory can be exported from master instance terminal by adding OpenFOAM directory to the /etc/exports file as superuser, exporting this file and starting the NFS server:

```
sudo sh -c "echo '/home/ubuntu/OpenFOAM *(rw,sync,no_subtree_check)' \
        >> /etc/exports"
sudo exportfs -ra
sudo service nfs-kernel-server start
```

The next step is to delete empty directories in the OpenFOAM directory on each slave instance and use that directory as a mount point for mounting the OpenFOAM directory on the master instance. For this user needs private IP of the master instance and private IP of each slave instance. This can be partially automated by defining environment variable which will store private IPs of slave instances. It has to be done each time instance is started.

```
VAR="ip_slave1 ip_slave2 ip_slave3 ..."
for IP in $VAR ; do ssh $IP 'rm -rf ${HOME}/OpenFOAM/*' ; done
for IP in $VAR ; do ssh $IP 'sudo mount ip_master:${HOME}/OpenFOAM \
        ${HOME}/OpenFOAM' ; done
```

Mounting of the OpenFOAM directory can be tested for each slave by the command:

```
for IP in $VAR ; do ssh $IP 'ls ${HOME}/OpenFOAM' ; done
```

It should return *ubuntu-<version>* directory.

2.3.2 Running test case

Cluster will be tested on the damBreak tutorial case. It will involve changing to $FOAM_RUN directory, copying case from the *tutorials* directory, generating a mesh, refining the mesh, creating field file and initialising with *setFields*. Then mesh will be decomposed using *decomposePar* utility and running *interFoam* solver in parallel. First log onto the master node:

```
ssh -A ubuntu@master_ip
```

Then prepare the simulation by executing following commands:

```
run
cp -r $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak/damBreak .
cd damBreak
blockMesh
refineMesh -overwrite
cp -r 0/alpha.water.orig 0/alpha.water
setFields
decomposePar
```

Note: number of subdomains is specified in *decomposeParDict* file in *system* directory. User needs to specify a list of host machines to run case parallel on a cluster. It should be specified in *machines* file in case directory. File content should look like this:

```
ip_host1
ip_host2
ip_host3
.
.
.
```

Case can be run in parallel on a number of cores specified by *processor* directories using the host names in the *machines* file using *foamJob* script with -p flag:

```
foamJob -p interFoam
```

NOTE: author's tests showed that it is better to disable hyperthreading during creation of Instances, because it gives no improvement in calculation time.

3 AWS CLI

Note: this section is based on AWS CLI (Command Line Interface) documentation which can be found at AWS website.

3.1 Installing AWS CLI on Linux

According to the AWS CLI documentation the best way to assure that the newest version is installed, user should install AWS CLI via *pip*. The process of installing *pip* if it is not present on the system will not be described as it can be found easily on the web. AWS CLI can be installed by the command:

```
pip install awscli --upgrade --user
```

–upgrade updates all requirements and *–user* option installs the program to a subdirectory of user directory.

3.2 Configuring AWS CLI

The fastest way to start is to execute *aws configure* command. This will add default profile, which will be used for executing AWS command if no profile will be explicitly specified. User must specify AWS Access Key ID, AWS Secret Access Key, default region name and default output format. *AWS Access Key* and *AWS Secret Access Key* are connected with AWS credentials connected with IAM user or role. *Access Key ID* and *Secret Access Key* can be created via IAM console in *Security credentials* tab. *Default region name* specifies appropriate server. *Default output format* specifies format of the results (options include: json, text, table).

Other profiles can be created by executing *aws configure* with *–profile* flag:

```
aws configure --profile user2
```

Then, commands can be run via default profile, or specified by aforementioned flag. I.e.

```
aws s3 ls
aws s3 ls --profile myuser
```

Credentials data can be stored in many places on the computer and the precedence is defined in the following order:

1. Command line options, i.e. *–region, –output, –profile* flags.
2. Environment variables: *AWS_ACCESS_KEY_ID*, *AWS_SECRET_ACCESS_KEY*, *AWS_SESSION_TOKEN*.

3. CLI credentials file located at */.aws/credentials*.
4. CLI configuration file located at */.aws/config*.
5. Container credentials.
6. Instance profile credentials.

Using environment variables

Non-default profile can be used for single command as described above, but for multiple commands it can be tedious. Hence, one can create temporary environment variable (lasting until the end of shell session) by executing following command:

```
export AWS_PROFILE=profile_name
```

Additionally, another environment variables can be set to change options for shell process. List of available commands can be found in AWS Documentation, section Environment Variables.

Command Completion

AWS CLI includes command-completion feature which can be used with the **TAB** key. It might not be installed automatically. First, start by locating shell's installation directory by running:

```
echo $SHELL
```

Then, locate AWS Completer by running:

```
which aws_completer
```

Depending on shell type, different command must be used to enable AWS Completer. In case of bash, add

```
complete -C 'path_to_aws_completer/aws_completer' aws
```

into */.bashrc* file if you want it to be enabled every time you launch the shell.

4 Running EC2 Instance via AWS CLI[8]

First of all, install AWS CLI as described in section 3.1. Then, run *aws configure* and set up your credentials. Verify that credentials are configured correctly by executing:

```
aws ec2 describe-regions --output table
```

It should return table containing list of regions available at AWS. The next step is to create a Security Group and Key Pair. As mentioned in section 2.3, rule must be added to allow connection via SSH tunnel. Note that *–vpc-id* parameter will be omitted as default VPC will be used. Run

```
aws ec2 create-security-group --group-name tutorial-sg \
 --description "security group for tutorial"
```

It should return security group ID, which will be needed when launching the instance. Set described permission with following command:

```
aws ec2 authorize-security-group-ingress --group-name tutorial-sg \
        --protocol tcp --port 22 --cidr 0.0.0.0/0
```

Next, create key pair to be able to connect to the instance:

```
aws ec2 create-key-pair --key-name tutorial-key --query 'KeyMaterial' \
        --output text > tutorial-key.pem
```

[8] AWS, URL: https://docs.aws.amazon.com/cli/latest/userguide/tutorial-ec2-ubuntu.html, Accessed: 12.12.2018

Do not forget about changing permissions for the key file:

```
chmod 400 key_name.pem
```

Now, you are ready to launch and connect to the instance. You will need –image-id parameter which specifies AMI used. In case of OpenFOAM 6 on Ubuntu 18.04 provided by CFD Direct it is ami-e0d2d10b. Other parameters are self-explanatory. This command returns instance ID.

```
aws ec2 run-instances --image-id ami-e0d2d10b --security-group-ids \
        <security-group-id> --count <instances_number> --instance-type \
        <i.e. t2.micro> --key-name tutorial-key \
        --query 'Instances[0].InstanceId'
```

When instance is running, you can query public ip address, so you can connect to this instance.

```
aws ec2 describe-instances --instance-ids <instance ID> \
        --query 'Reservations[0].Instances[0].PublicIpAddress'
```

Now you can connect via ssh to your Instance.

4.1 CLI Skeleton[9]

In order to speed up creation of instances, CLI Skeleton can be prepared. CLI Skeleton in JSON file which stores parameters of *aws ec2 run-instance* command. User can specify any combination of parameters within file. This file template can be generated by running command (and saved to file):

```
aws ec2 run-instances --generate-cli-skeleton > file
```

After creating this template, it can be tailored to user's needs by editing in any text editor. Note that when for example KeyName is specified, user does not specify extension of file, rather just name of key. As most often the same set of parameters will be specified during creation of instances, CLI Skeleton will be used for creating OpenFOAM cluster via CLI. Useful parameter for template testing purposes is *DryRun* as when set to *true* it checks whether JSON is formed correctly.

5 Script for cluster creation

The attached script will create appropriate IAM role, instance profile, placement group, security group, then run specified instances, copy data to these instances, run selected script with OpenFOAM instructions, send data to S3 bucket and finally, terminate all instances connected with the case. Here only description can be found, full script can be found at GitHub.

First, variables containing names and paths are specified. The following list explains meaning of non self-explanatory variables:

- *caseName* determines name of the folder where data will be stored as well as tag for instances and names for IAM policies, roles and instance profile,
- *instanceType* is self-explanatory, however, user must bear in mind that not all instance types allow for certain options used in .json templates, especially specifying CPU options,
- *numberOfCPUs* can be specified if instance type allows, so user can use less physical processors than default for instance type,
- *numberOfNodes* means how many instances will be created,
- *openFoamScript* specifies path to the script containing OpenFOAM connected commands which shall be executed on master node,

[9] AWS, URL: https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-skeleton.html, Accessed: 14.12.2018

- *masterCliSkeleton, slaveCliSkeleton* specifies path to templates for starting instances,
- *S3Bucket* stores path to S3 folder, i.e. xxx/xyz/, where xxx is bucket name and xyz is folder within bucket,
- *EC2trustFile* stores path to file containing JSON data about policy allowing EC2 instance to assume role,
- *permissionsPolicyFile* stores path to JSON file with IAM policy specification.

Then, derived variables are created containing information about instances names, IAM names and about temporary files. Next, specified files are checked, i.e. whether they exist and are readable and if instance type is correct. In the next step, temporary files are edited to reflect user specified parameters. Important information for IAM policies is an account ID. In this script it is acquired by aws sts get-caller-identity command.

```
userID=$(aws sts get-caller-identity --query 'Account' --output text)
```

In the following section, instance profile will be created, however, there are some steps needed to be taken before to ensure no errors. First of all, if instance profile with given name exists, it must be deleted. In order to achieve this, all roles have to be removed from instance profile. Then, it can be deleted. The next step is to remove old IAM role, which has the same name as the one user wants to create. IAM role cannot be deleted if it has policies attached, so firstly policies are detached and then role is deleted. Now, new role can be created with policy specifying that EC2 service instances can assume role. Then, temporary file is created, so that S3 bucket etc. can be specified prior to creating new instances. Next, policy is attached to the role, which enables sending data to S3 bucket and terminating instances with given resource tag. Finally, role is added to instance profile and instance profile is associated with master instance CLI-skeleton. Due to the time needed for AWS to coordinate data across their services, sleep command was added as without this wait, instance creation signal was sent before instance profile was ready, which lead to and error and lack of creation of master instance. Finally, instances are created and temporary files are removed. Agent forwarding shall be enabled so the user can communicate with slave instances via master. Then, one should wait for initialization of instances to be finished. Then, public ip of master node must be gathered in order to be able to connect via ssh. In addition, private ip numbers of master node and all slaves are downloaded. These are needed for NFS server connection and for mpi library later used in OpenFOAM parallel calculations. In the next step, script containing OpenFOAM commands is copied, however, if simulation case is not a variation of tutorial case, i.e. it can be copied from tutorial with minor modification, user should also copy necessary files at this point. The purpose of this loop is to ensure files are really copied, as it happened during testing that even though the state of instances was "running", ssh connection failed. Hence the double checking. The next step is to create NFS server, so all data can be stored at master node, which enables slave disk to be small and hence saving some credits. One of the steps is to create known host file at master node, so all the ssh connection can be done in background without prompting user for input. All of the following ssh commands could be sent simultaneously. Additionally, it is necessary to create the list of private ip numbers for mpi library. One last piece of information must be fetched from AWS, namely IDs of all instances with given tag, so all nodes will be terminated after end of calculations. Finally, the set of commands are sent to master node with nohup command, so these commands will be executed independently from user's machine state. It launches OF script, waits for it to finish, then sends data to S3 bucket and terminates all instances. Meanwhile, at user's computer command is sent to AWS to revoke security group rule allowing connection via ssh to master node, making cluster unreachable and safe. If user needs to connect later to master node, i.e. to check if calculations are being done, new inbound rule must be created as it was described already.