Comparison with Docker Swarm

This lesson is a comparison between Kubernetes Operations (kops) and Docker for AWS.

WE'LL COVER THE FOLLOWING

The Similarities

Docker For AWS (D4AWS) quickly became the preferable way to create a Docker Swarm cluster in AWS (and Azure). Similarly, kops is the most commonly used tool to create Kubernetes clusters in AWS.

The result, with both tools, is more or less the same.

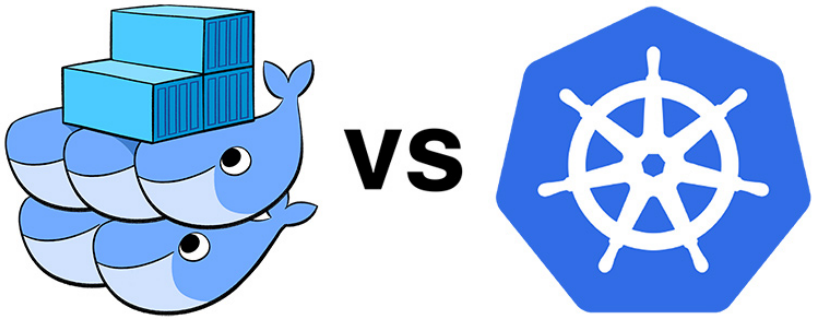
- Both create Security Groups, VPCs, Auto-Scaling Groups, Elastic Load Balancers, and everything else a cluster needs.
- In both cases, Auto-Scaling Groups are in charge of creating EC2 instances. Both rely on external storage to keep the state of the cluster (kops in S3 and D4AWS in DynamoDB).
- In both cases, EC2 instances brought to life by Auto-Scaling Groups know how to run system-level services and join the cluster.

If we exclude the fact that one solution runs Docker Swarm and that the other uses Kubernetes, there is no significant functional difference if we observe only the result (the cluster). So, we'll focus on user experience instead.

The Differences

The Differences

Both tools can be executed from the command line and that's where we can spot the first difference.



Docker

Docker for AWS relies on CloudFormation templates, so we need to execute `aws cloudformation` command. Docker provides a template and we should use parameters to customize it. In my humble opinion, the way CloudFormation expects us to pass parameters is just silly.

Let's take a look at an example.

```
aws cloudformation create-stack \
    --template-url https://editions-us-east-1.s3.amazonaws.com/aws/stable/Docker.tmpl \
    --capabilities CAPABILITY_IAM \
    --stack-name devops22 \
    --parameters \
    ParameterKey=ManagerSize,ParameterValue=3 \
    ParameterKey=ClusterSize,ParameterValue=2 \
    ParameterKey=KeyName,ParameterValue=workshop \
    ParameterKey=EnableSystemPrune,ParameterValue=yes \
    ParameterKey=EnableCloudWatchLogs,ParameterValue=no \
    ParameterKey=EnableCloudStorEfs,ParameterValue=yes \
    ParameterKey=ManagerInstanceType,ParameterValue=t2.small \
    ParameterKey=InstanceType,ParameterValue=t2.small
```

Having to write something like `ParameterKey=ManagerSize,ParameterValue=3` instead of `ManagerSize=3` is annoying at best.

Kubernetes

A sample command that creates Kubernetes cluster using kops is as follows.

```
kops create cluster \
    --name $NAME \
    --master-count 3 \
    --node-count 1 \
    --node-size t2.small \
    --master-size t2.small \
    --zones $ZONES \
    --master-zones $ZONES \
    --ssh-public-key devops23.pub \
    --networking kubenet \
    --kubernetes-version v1.14.8 \
    --yes
```

Isn't that easier and more intuitive?

Moreover, kops is a binary with everything we would expect. We can, for example, execute `kops --help` and see the available options and a few examples. If we'd like to know which parameters are available with Docker For AWS, we'd need to go through the template. That's definitely less intuitive and more difficult than running `kops create cluster --help`. Even if we don't mind browsing through the Docker For AWS template, we still don't have examples at hand (from the command line, not browser).

Concluding Remarks

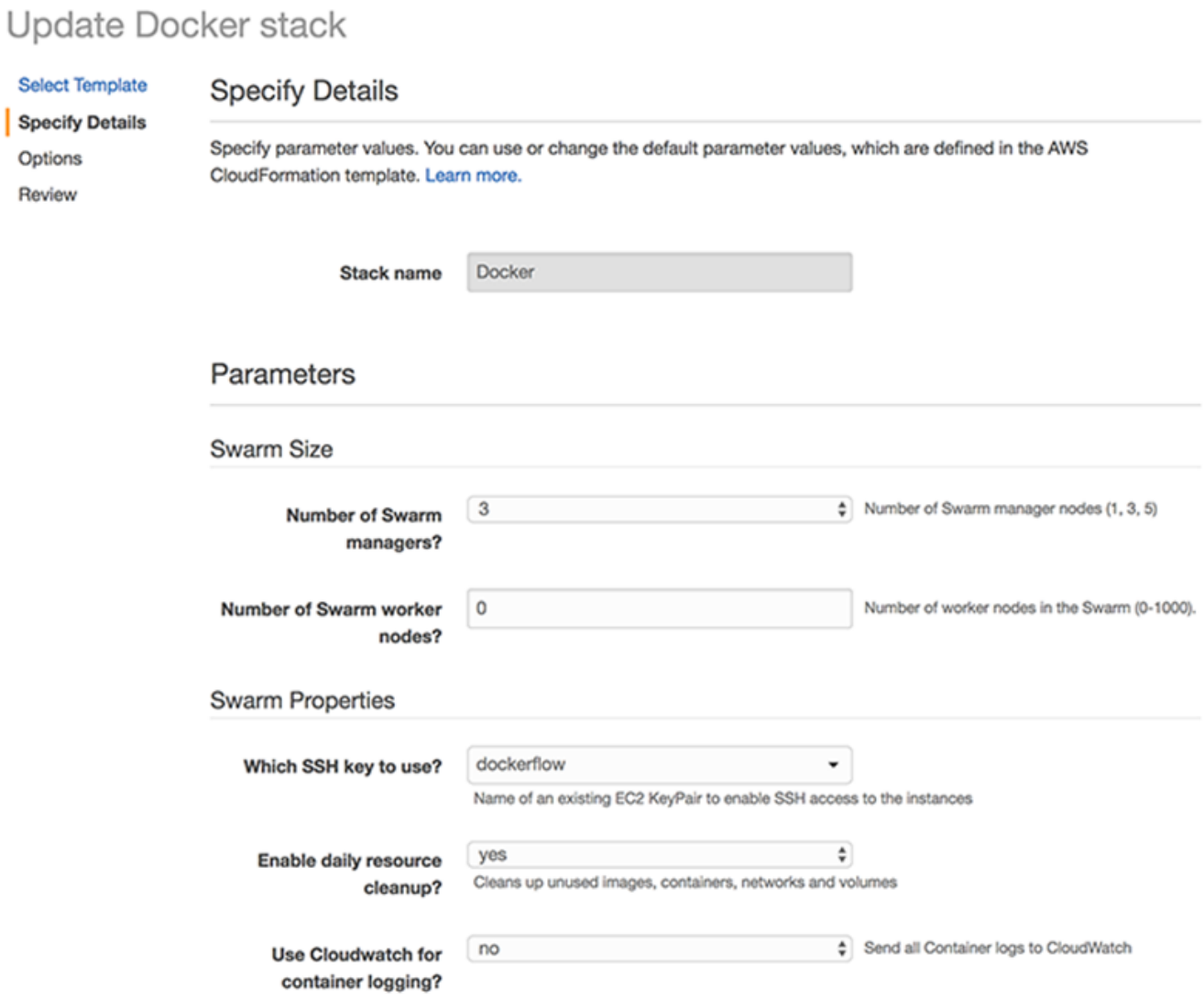
From a user experience perspective, **kops wins** over Docker For AWS if we restrict the comparison only to command line interface. Simply put, executing a well-defined binary dedicated to managing a cluster is better than executing `aws cloudformation` commands with remote templates.

Did Docker make a mistake for choosing CloudFormation? We don't think so. Even if command line experience is suboptimal, it is apparent that they wanted to provide an experience native to hosting vendor. In our case that's AWS, but the same can be said for Azure. If you will always operate cluster from the command line (as I think you should), this is where the story ends and kops is a winner with a very narrow margin.

The fact that we can create Docker For AWS cluster using CloudFormation means that we can take advantage of it from AWS Console. That translates into UI experience. We can use AWS Console UI to create, update, or delete a cluster. We can see the events as they progress, explore the resources that were created, roll back to the previous version, and so on. By choosing CloudFormation template, Docker decided to provide not only command line but also a visual experience.

but also a visual experience.

One may think that UIs are evil and that we should do everything from the command line. That being said, we're fully aware that not everybody feels the same. Even if you do choose never to use UI for "real" work, it is very helpful, at least at the beginning, as a learning experience of what can and what cannot be done, and how all the steps tie together.



Docker For AWS UI

It's a tough call. What matters is that both tools are creating reliable clusters. Kops is more user-friendly from the command line, but it has no UI. Docker For AWS, on the other hand, works as a native AWS solution through CloudFormation. That gives it the UI, but at the cost of the suboptimal command line experience.

You won't have to choose one over the other since the choice will not depend on which one you like more, but whether you want to use Docker Swarm or Kubernetes.

In the next chapter, we will focus on achieving state persistence using AWS persistent volumes.