AWS Exploration

There are many different options available to run containers on AWS, so we would like to determine the best way to run Clowder in containers on AWS.

Scenario A: Docker Compose via EC2 - Identical to what we do on OpenStack

Overview

The most familiar option would be to use AWS EC2 to spin up one or more raw virtual machines, precisely how we do on OpenStack today.

We would just need to install Docker on the host, and use Clowder's docker-compose.yml to bring up the core pieces of the Clowder stack.

After deployment we would maintain the individual EC2 VMs that run the cluster components, while Docker Compose runs the containers and restarts them if they hit an error.

The compose file can be edited to the user's liking to provide the desired set of configuration options / extractors needed.

The main downside here would likely be long-term cost - while we likely wouldn't need any additional learning to get started, there may be more AWS-friendly ways to save money on the long-running infrastructure.

Pros

- No additional learning required

Cons

- Price of this option could prove to be higher than others

Example

https://opensource.ncsa.illinois.edu/bitbucket/projects/CATS/repos/clowder/browse/docker-compose.yml

Pricing

- Hourly pricing depends on chosen instance sizes
    - I think that "On Demand" is the most basic/common setup
    - See EC2 On Demand instance pricing: https://aws.amazon.com/ec2/pricing/on-demand/
- See EC2 Pricing: https://aws.amazon.com/ec2/pricing/

Open Questions

- Prices? What do we know about our expected workload?
    - We could do some CPU/memory/IO profiling on existing systems
- Are there better options out there that could bring down price without too much additional architecture?

Scenario B: Kubernetes via EC2 (kops) - Classic Kubernetes with an AWS twist

Overview

AWS has a tool called kops that can be used to deploy several EC2 instances and install the necessary services for running them as a Kubernetes cluster.

We would just need to spin up an EC2 instance that can house our deployment/configuration data, then use `kops` from that machine to provision master /worker VMs and bring up a cluster.

The compose (mentioned in scenario A above) can be edited to the user's liking to provide the desired set of configuration options / extractors needed.

After deployment AWS will monitor and we would maintain the individual EC2 VMs that run the cluster components, while Kubernetes manages the containers and keeps them all healthy and running.

We also maintain control over the master node(s) in the case, and are free to modify them as we wish, but we would then be footing the bill for the master node(s) and need to maintain/monitor them to keep them running.

Autoscaling components can be deployed to keep your cluster size to a minimum when there is less load, and can automatically increase the size of the cluster by spinning up new loads once a bottleneck is hit.

Pros

- Maximum flexibility / control over your cluster

Cons

- Most complex to learn/setup (but I've done it before)

Example

KnowEnG Kubernetes Deployment

Pricing

- Hourly pricing depends on chosen instance sizes
    - I think that "On Demand" is the most basic/common setup
    - See EC2 On Demand instance pricing: https://aws.amazon.com/ec2/pricing/on-demand/
- Autoscaling is available and works fairly well given reasonable constraints
- See EC2 Pricing: https://aws.amazon.com/ec2/pricing/

Open Questions

- Prices? What do we know about our expected workload?
    - We could do some CPU/memory/IO profiling on existing systems

Scenario C: Kubernetes via EKS - Kubernetes where AWS manages your master nodes?

Overview

AWS has a newer service called EKS (Elastic Kubernetes Service) that can be used to deploy several EC2 instances as a Kubernetes cluster.

We would just need to use the AWS CLI to create a new EKS cluster - this will, like `kops`, spin up a set of EC2 instances and provision them as a Kubernetes cluster.

The compose (mentioned in scenario A above) can be edited to the user's liking to provide the desired set of configuration options / extractors needed.

After deployment AWS will monitor and we would maintain the individual EC2 VMs that run the cluster components, while Kubernetes manages the containers and keeps them all healthy and running.

We lose control over the master node(s) in the scenario (versus Scenario B, above), but this may save money in the long-term.

Autoscaling components can be deployed to keep your cluster size to a minimum when there is less load, and can automatically increase the size of the cluster by spinning up new loads once a bottleneck is hit.

Pros

- Less infrastructure to manage, and therefore less cost associated with infrastructure

Cons

- Assumption: we may have less control over what gets deployed (kube-apiserver flags, K8S versions, etc)

Example

https://github.com/aws-samples/aws-workshop-for-kubernetes

Pricing

- "You pay $0.20 per hour for each Amazon EKS cluster that you create."
- "You can use a single Amazon EKS cluster to run multiple applications by taking advantage of Kubernetes namespaces and IAM security policies."
- "You pay for AWS resources (e.g. EC2 instances or EBS volumes) you create to run your Kubernetes worker nodes."
- "You only pay for what you use, as you use it; there are no minimum fees and no upfront commitments."
- See EKS Pricing: https://aws.amazon.com/eks/pricing/

Open Questions

- Prices? What do we know about our expected workload?
    - We could do some CPU/memory/IO profiling on existing systems

Scenario D: ECS via FarGate - Containers, minus the Infrastructure?

Overview

AWS has an even newer service called FarGate that can be used to deploy containers directly to the cloud without worrying about the machines they will run on.

In this scenario, we pass the docker-compose.yml of Clowder or whichever other services we want FarGate to run.

FarGate then figures out where to run it and charges us for the memory/CPU usage of our containers while it runs.

While we likely could test out this scenario immediately without any/many modification to Clowder's docker-compose file, it may end up being more expensive in the long-term.

Pros

- Least effort to manage long-term - no underlying infrastructure to worry about, and supposedly pay based on usage
- Easiest migration, as it appears to take a format similar to docker-compose

Cons

- Assumption: Least flexible - may not be able to run all types of workloads here, but we shouldn't need to run cronjobs or batch jobs or anything like that
- Many unknowns - it's unclear what potential features we might be losing here (for example, LMA)

Example

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-cli-tutorial-fargate.html

Pricing

- "The price per vCPU is $0.00001406 per second ($0.0506 per hour) and per GB memory is $0.00000353 per second ($0.0127 per hour)."
- "You pay only for what you use."
- See FarGate Pricing: https://aws.amazon.com/fargate/pricing/

NOTE: Some users say that FarGate is better suited for batch operations, and not for services that are expected to run 24/7

Open Questions

- Prices? What do we know about our expected workload?
    - We could do some CPU/memory/IO profiling on existing systems
- LMA: do we need to provide monitoring/logging/alerts ourselves? Perhaps this is something that AWS FarGate provides for its containers?
- Persistent storage: are Docker volumes safely backed by EBS, EFS, or some other persistent storage options?
    - What are the costs involved with this storage?
    - Does this give us any additional S3 support?
- Shared storage: do we need shared storage (e.g. NFS/EFS)? Is such a thing provided by FarGate?
    - What are the costs involved with this storage?
- Networking: How do the container networks work? Does it simply use the configured networks in the docker-compose.yaml as expected?

Scenario E...? ECS via EC2 - Containers, minus the familiar parts of the Infrastructure?

Overview

Before FarGate, the ECS option offered by ECS was center around EC2 and their custom taskDefinitions.

You first need to spin up an ECS cluster (using the UI or CLI). Your cluster size determines how many tasks you can run concurrently.

You can then define your tasks using a special AWS-flavored JSON/YAML syntax, then submit it to the ECS to run your tasks in containers.

This scenario appears to be the most work with less (or less obvious) methods for cutting costs.

Pros

- ?? (no obvious advantage yet over say FarGate)

Cons

- Task definitions appear to be highly customized and likely AWS-specific

Example

https://aws.amazon.com/getting-started/tutorials/deploy-docker-containers/

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS_GetStarted_EC2.html

Pricing

Unclear, but I suspect that EC2 Instance Pricing may apply

Open Questions

- Why would we even consider this over FarGate, especially if our application already leverages Docker Compose? Is it somehow cheaper in the long-term?