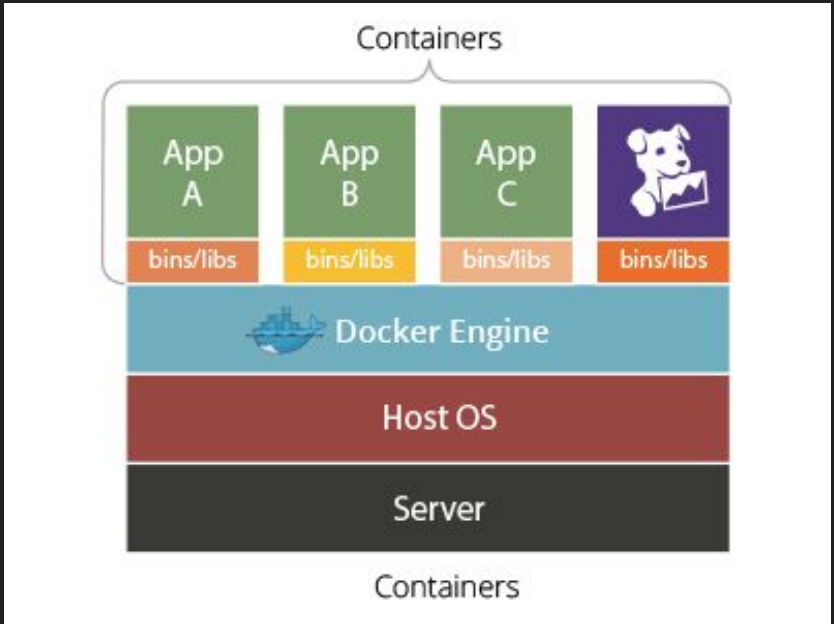
ECS - Amazon ES2 Container Service

Running Docker enabled-apps in EC2 clusters.

Docker

Containers Overview.



- Self Contained.
- Portable.
- Easy deployment.
- Development life cycle gets simplified.
- Small footprint.
- Decoupling big systems.
- Reduces library complexity.
- Efficient management of resources.

A container pipeline

- OPS team creates certified base images with operating system.
- Images are turned into utilities containers (Redis, Nodejs, MongoDB).
- Developers use these containers to build source code.
- Version control.
- Build & unit tests.
- Automated acceptance tests.
- User acceptance tests.
- Release.

Scaling up: Challenges.

Docker is powerful when you run containers on your laptop or server, but now we need to scale up to hundreds or thousands of containers.

- Implementing our own cluster will add more complexity to the overall system.
- We need to know what hosts have the appropriate resources.
- We need to know the state of our system.
- Visibility. What if the container has issues or dies?.
- More complexity, more management = Potential disaster.

We need a solution to take care of this issue and focus on our product.

Amazon ECS

What | Why | How

Amazon ECS | What.

- Amazon EC2 Container Service is a highly scalable, high performance container management service that supports Docker containers and allows you to run applications on a cluster of Amazon EC2 instances.

In other words

ITS AWESOME.

Amazon ECS | Why.

- Eliminates (reduces) the need to install, operate, and scale our own cluster management infrastructure.
- Maintains application availability better than other solutions.
- Scale up or down to meet your application's requirements.
- Integration with Elastic Load Balancing.
- Multiple containers per host can use the same port.

Amazon ECS | Why.

- Possibility to use of your own schedulers.
- Integrate with your existing software delivery process (CI/CD).
- Powerful monitoring [CloudWatch]
- AWS/ECS CLI access.

Amazon ECS | How.

The following is an overview of the methodology used to implement an ECS cluster to run docker containers.

For detailed information check documentation on
https://github.com/acanessa/good-uncle-ecs

Prerequisites

- General understanding of AWS and Docker
- Access to EC2 instances, Elastic Load Balancer, Credentials, Docker Hub.
- Coffee. Lots of coffee.

Demo

DEMO - General overview

The first step is to create an empty ECS cluster. This cluster will be populated with EC2 instances that will run the docker containers via the ECS Agent and Docker Engine. We need to configure the instances with the ECS Agent, Docker Engine and IAM role. The instances will be now part of the cluster.

We define the properties of the docker containers via task-definitions, later they will be used in the service that we wish to run in the cluster.

When we deploy the task (a task is an instance of a task-definition) ECS will pull the docker image from docker hub and deploy it according to configuration.

DEMO

If the EC2 instance should crash, ECS will automatically run the docker container in the second instance, cluster orchestration is done for us!

In this demo, the container is using port 8080. We can’t run another container with the same port in the EC2 host so we implement an application-type Elastic Load Balancer, and map dynamic host ports to the 8080 container port.

EBL routes requests from its internet-facing port 80, via VPC firewall, to the dynamic port that will subsequently, be routed to the port 8080 of the container.

The result? We can now run multiple containers with the same port in one host.

DEMO

The demo has 4 containers running in one service. The container is a simple node app that will show the containerID and save it into a MongoDB database.

URL: http://elb-1-2084531530.us-west-2.elb.amazonaws.com/

RESULT: "container-id": "098d5496d8a5"

When we refresh the browser, containerID will change indicating that the ELB is routing requests (as load in all containers is basically the same) in a round-robin fashion.

DEMO

- Should one of the containers fail, the other three would still be accepting requests and saving data to the MongoDB database.
- Should demand increases, more containers can be created using auto-scaling. Same concept if demand decreases.
- Should one of the EC2 instances fail (rare), the cluster will recreate the containers in the other instance (subject to resource availability)

Can we do more?

Yes!.

A few concepts to try are:

- Stress testing to see how ECS manages auto-scaling.
- Extracting metrics via AWS/ECS CLI.
- Using these metrics with an ELK stack for visualization.
- Implementing own scheduler via AWS/ECS CLI.
- Adding or removing EC2 instances dynamically.
- Integrating ECS with coffee machine to assure coffee delivery process.

Conclusion

Conclusion

Using Amazon Elastic Container Service would allow us to scale our service without the usual problems of implementing and managing our own cluster infrastructure while at the same time, will give us the possibility to integrate several AWS technologies.

Deployment time, management and costs will be drastically reduced. Service availability and quality will be increased.

In short, a technology worthy to consider!

Thank you.