Deploying Containers to the Cloud

A review of some options and considerations for container deployments in public cloud environments

About me

- Husband, father, Jeeper, all-around geek
- Blogger (12 years at http://blog.scottlowe.org)
- Author (7 books so far, 2 more in the works)
- Speaker (Interop, VMworld, DevOps Networking Forum, OpenStack Summit, local meetups)
- Podcaster (The Full Stack Journey podcast)
- Employee at VMware, Inc., working on NSX and Cross-Cloud Services

About this session

- Intended to provide high-level view of public cloud deployment options for containers
- **NOT** intended to cover all possible options or variations
- Will cover these scenarios, followed by a live demo of each:
  - Docker Swarm on AWS
  - AWS Elastic Container Service (ECS)
  - Kubernetes on AWS
  - Google Container Engine (GKE; runs Kubernetes)

Before we start

- A PDF version of these slides will be available after the event
- There's a GitHub repository that contains files to allow you to replicate all demos (https://github.com/lowescott/2017-itx-container-workshop)
- Please exercise common courtesy (silence mobile devices, step outside if you need to take a call, etc.)

Prerequisites

Before you deploy containers to the cloud...are you prepared?

- Do you have policies and a process for reproducible builds?
- Is your organization ready for containers and container orchestration?
- Have you addressed how you'll secure containers and container images?

Guaranteeing reproducible builds

- Controlled sources for all packages
- No use of `:latest` for building Docker images (specific versions)
- Clear understanding of how base layers are built (be sure to review base layer's `Dockerfile` to understand dependencies)
- Official images are not exempt from review—see official MariaDB image `Dockerfile` for an example

Reproducible builds: Live demo

Tools: Docker Hub, GitHub

Organizational readiness

- Who will own container images?
- Does your staff have the skills to support container orchestration frameworks?
- How’s the interaction between your developers and your operations team(s)?
- Are your developers ready for microservices?
- This isn’t just a technology issue—you also need to address organizational/cultural issues

Container security

- Everything mentioned earlier about reproducible builds also applies here
- Most orchestration frameworks haven't yet defined mature mechanisms for providing network-based security controls for containers
- What about auditing and compliance? Ensuring consistent security policy between public cloud workloads and on-premises workloads?

TL;DR: There's a pretty fair amount of non-technical work that needs to be done before you start deploying containers to the cloud.

Option 1: Docker Swarm on AWS EC2

Swarm on EC2: Overview

- This illustrates a scenario where you're manually deploying/managing the container orchestration framework
- Leverages EC2 instances with Docker CE and Swarm mode
- Uses "standard" Docker concepts (containers, services, networks) and tools (like `docker-compose`)

Swarm on EC2: Technologies involved

- Docker CE (Community Engine): container runtime and daemon
- Swarm mode (part of Docker CE): orchestration layer
- Amazon EC2: Compute instances on which containers will run
- Terraform: IaaS orchestration tool for creating AWS infrastructure
- Ansible: Automation/configuration management tool for configuring instances

Swarm on EC2: Pros & cons

- **Pro:** Doesn’t leverage any cloud-specific features
- **Con:** Lacks strong integration to cloud-specific features
- **Pro:** Theoretically possible to port or extend to any cloud provider with minimal changes
- **Pro:** Can use “standard” Docker concepts and tools
- **Con:** “Locked in” to Docker's tools, runtime, and images
- **Con:** User/consumer responsible for managing the orchestration framework

Swarm on EC2: Live demo

Tools: Terraform, Ansible, Ubuntu Linux, Docker

Option 2: AWS Elastic Container Service (ECS)

ECS: Overview

- This an example of leveraging a provider-supplied orchestration framework
- ECS leverages EC2 instances w/ an ECS Agent installed
- Key building blocks are task definitions, tasks (containers) and services (declarative deployments of containers)

ECS: Technologies involved

- Amazon EC2: Compute instances on which containers will run
- Amazon RDS: Database-as-a-service (DBaaS) providing MySQL database for demo application
- Amazon ECS: Container orchestration solution responsible for scheduling containers to execute on EC2 instances
- CloudFormation: Orchestration tool to create AWS infrastructure/services

ECS: Pros & cons

- **Pro:** Doesn't require you (the consumer) to manage the orchestration framework; that's handled by AWS
- **Con:** Proprietary and specific to AWS
- **Con:** "Locked in" to AWS
- **Pro:** Tightly integrated with and leverages AWS-specific functionality

ECS: Live demo

Tools: CloudFormation, Amazon RDS, Amazon Linux

Option 3: Kubernetes on AWS

Kubernetes on AWS: Overview

- Another example of manually deploying and managing a container orchestration framework
- Uses Kubernetes on AWS
- Shows the use of a tool called **kops** to help automate the deployment of the Kubernetes cluster
- Key Kubernetes building blocks are pods (workload units), services (exposed endpoints for groups of pods), and deployments (declarative definitions of pods)

Kubernetes on AWS: Technologies involved

- Docker: Container runtime
- Kubernetes: Container orchestration layer
- Amazon EC2: Compute instances on which containers will run
- Amazon ELB: Load-balancing for containers/services
- Kops: Kubernetes-specific orchestration tool for turning up Kubernetes clusters

Kubernetes on AWS: Pros & cons

- **Con:** User/consumer responsible for deploying and managing the orchestration framework
- **Pro:** More (potential) flexibility than a provider-managed orchestration framework
- **Pro:** Theoretically possible to port or extend to another cloud provider with minimal changes
- **Con:** Works differently than "standard" Docker tools and uses different concepts

Kubernetes on AWS: Live demo

Tools: kops, Kubernetes, YAML definitions

Option 4: Google Container Engine (GKE)

GKE: Overview

- A bit of a unique case
- It’s Kubernetes, but hosted on Google Cloud and managed by Google
- Same provider-agnostic building blocks (because it’s Kubernetes), but without the user/consumer needing to manage the framework

GKE: Technologies involved

- Google Compute Engine (GCE): Compute instances on which containers will run
- GCE Load balancer: Providing load balancing for containers/services
- GCE persistent disks: Persistent storage for containers
- Google Container Engine (GKE): Hosted/managed Kubernetes orchestration layer

GKE: Pros & cons

- **Pro:** User/consumer doesn’t need to manage the orchestration framework (like ECS)
- **Pro:** Not necessarily tightly tied to Google because Kubernetes can run on other providers
- **Con:** Works differently than “standard” Docker tools and uses different concepts
- **Pro:** Can easily leverage Google Cloud features/services, if so desired

GKE: Live demo

Tools: GKE (Kubernetes), YAML definitions

Summary

- Provider-managed orchestration frameworks offload the management burden
- Consumer-managed frameworks offer more (potential) flexibility
- All options leverage Docker to some extent (some more than others)
- Kubernetes has both provider-managed and consumer-managed options, and supports multiple cloud platforms
- Docker Swarm has only consumer-managed options but does support multiple cloud platforms

Questions & answers

What questions do you have?

Thank you!

Scott Lowe

Blog: http://blog.scottlowe.org/
Twitter: https://twitter.com/scott_lowe
GitHub: https://github.com/lowescott
Life: Colossians 3:17