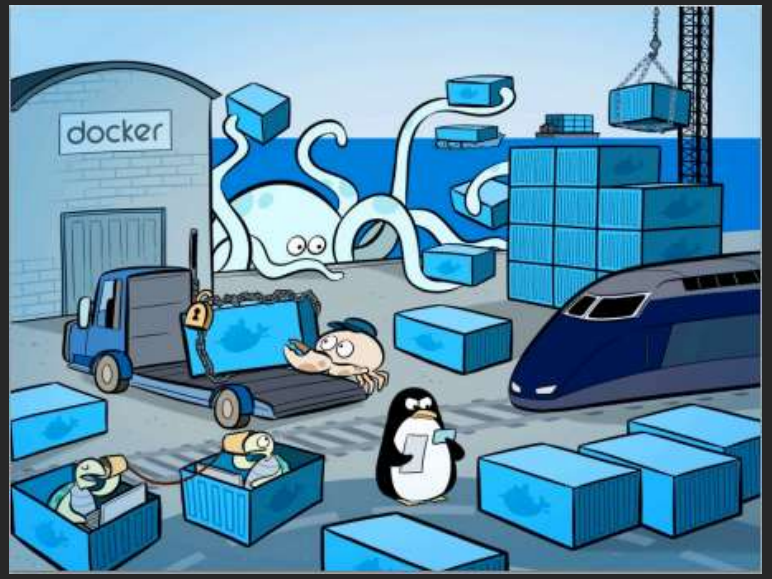
Docker on AWS - the Right Way

...

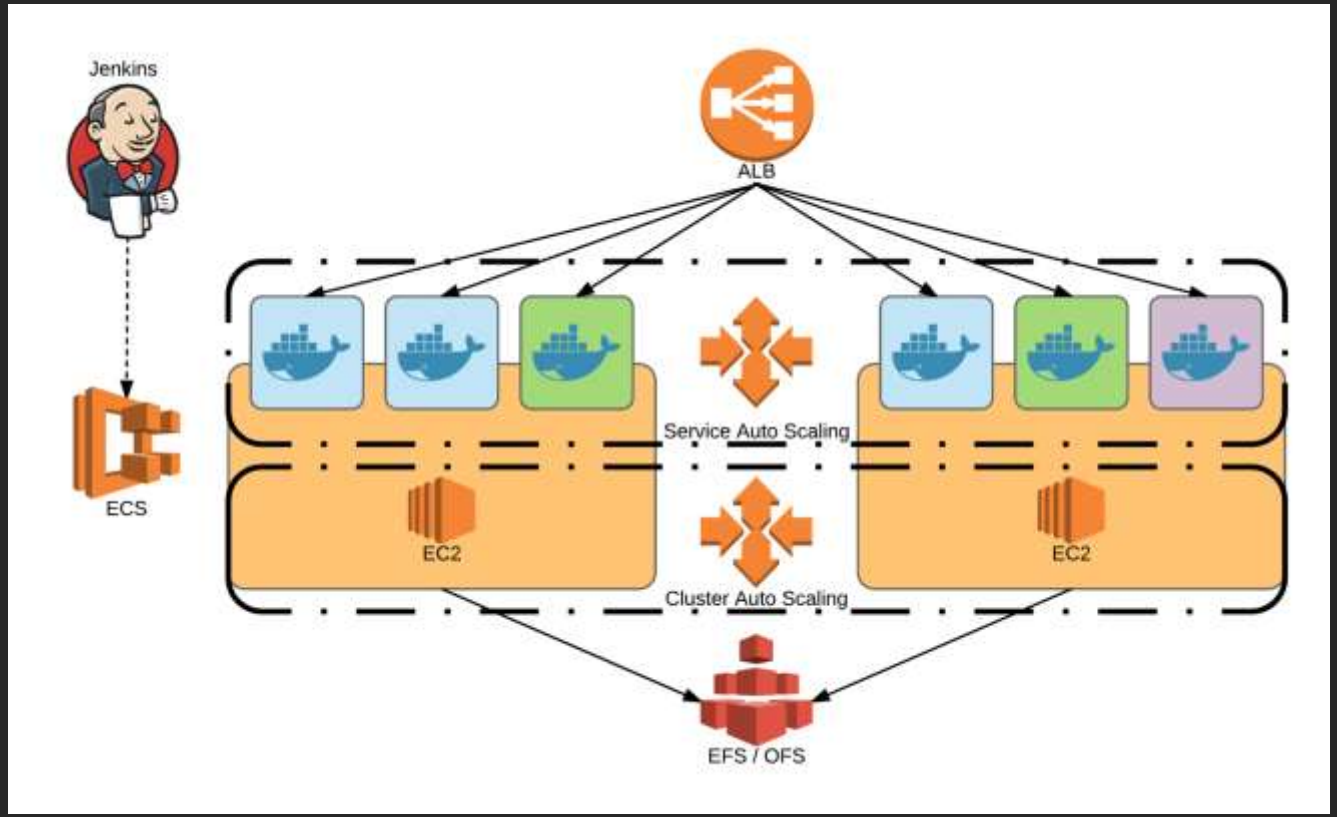
Johanan Lieberman

Agenda

- Container Orchestration on AWS
- Service Discovery
- Service Load Balancing
- Auto Scaling
- Storage
- Continuous Integration & Delivery

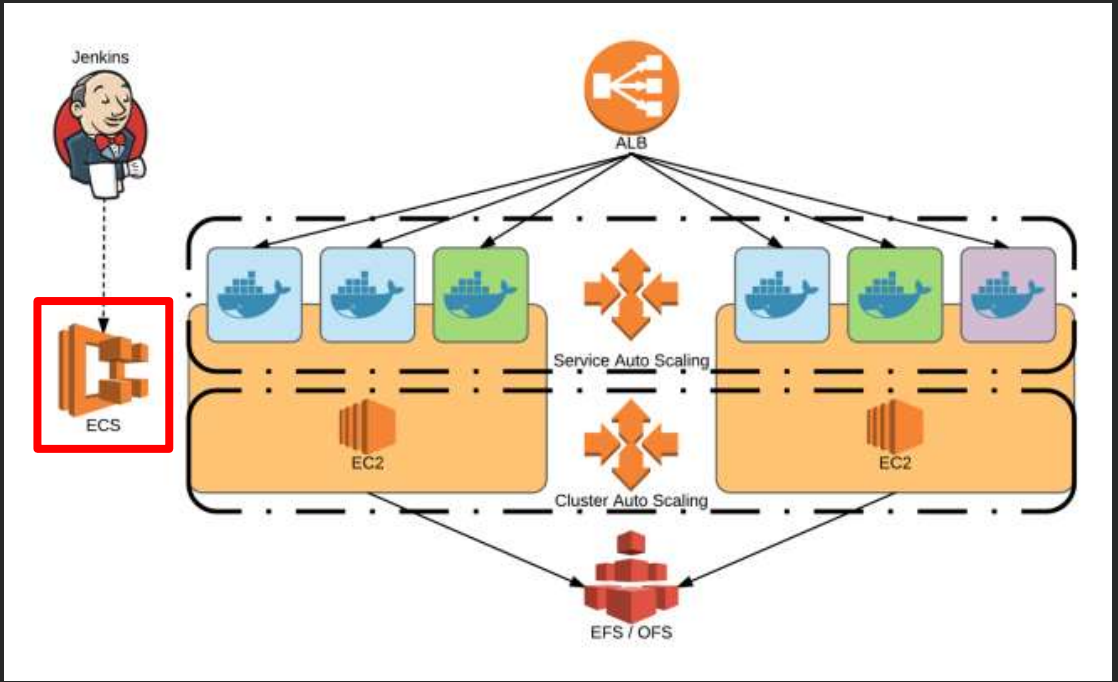


Reference Architecture



emind
YOUR CLOUD EXPERTS

Container Orchestration on AWS



What is Container Orchestration?

- Technologies which allow us to:
  - Create multi-node container clusters
  - Manage multiple containers easily
  - Automate container lifecycle

Why Do We Need Container Orchestration?

- Horizontal scalability across multiple hosts
- Grouping of related containers
- Automatic failure detection and recovery
- Seamless updates


CAUTION
9'6
HIGH
8 9:34

Why Do We Need Container Orchestration?

- Horizontal scalability across multiple hosts
- Grouping of related containers
- Automatic failure detection and recovery
- Seamless updates

ECS - EC2 Container Service

- Docker container orchestration service by AWS
- Operates on top of EC2
- Built-in private Docker registry (ECR)

Why Use ECS?

- Built-in security
  - Assign IAM Roles to Docker containers
  - Docker registry authentication using IAM
- Native integration with ELB and Auto Scaling
- Spot fleet + Auto Scaling support (announced Sep. 1, 2016)
- Full support from AWS

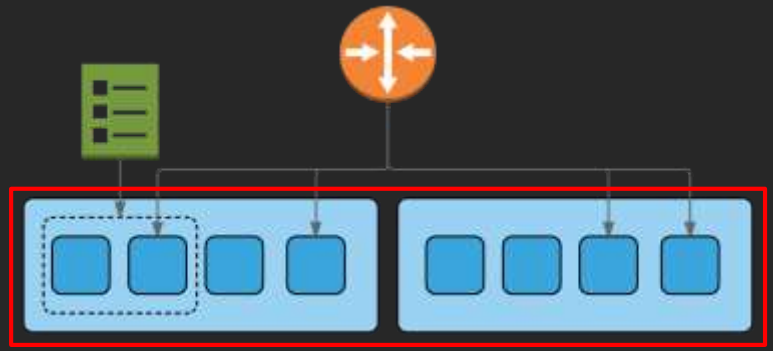
ECS Components

- **Cluster** - a group of container instances
- **Container Instance** - an EC2 instance that hosts containers
- **Task** - a set of related Docker containers
- **Task Definition** - a template which defines a task
- **Service** - a group of identical tasks

emind
YOUR CLOUD EXPERTS

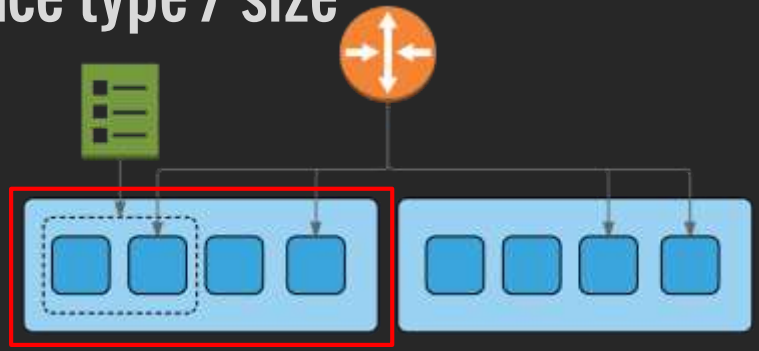
Cluster

- A group of container instances
- Supports multiple Availability Zones
- Bound to a specific AWS region



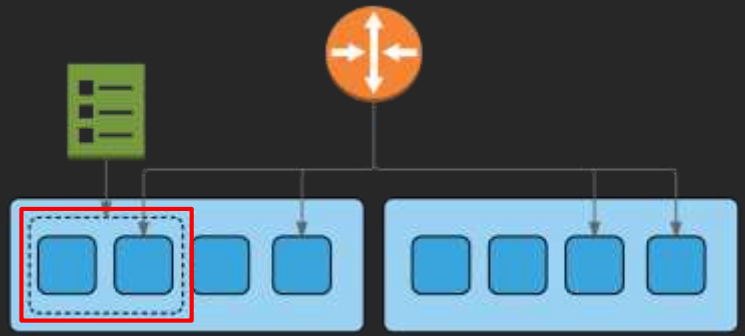
Container Instance

- An EC2 instance running Docker with an ECS agent
- May be deployed from an official AWS AMI
- May be deployed using an Auto Scaling group
- Can be of any EC2 instance type / size



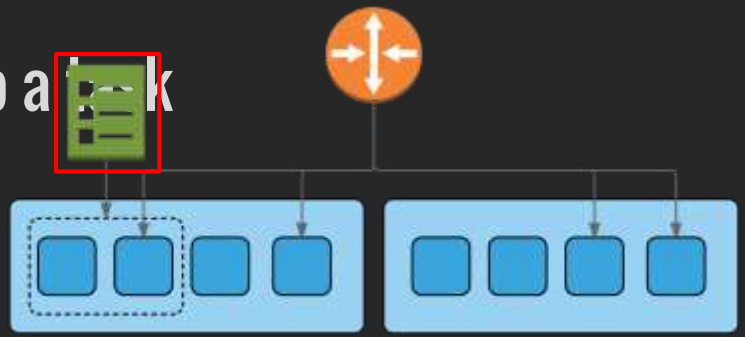
Task

- A set of one or more related containers
- Deployed to a cluster
- Containers within a task are placed on the same host



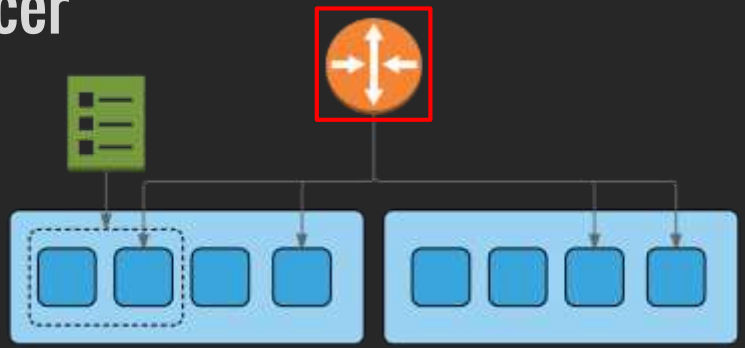
Task Definition

- Serves as a “template” for tasks
- Allows to define most of the Docker features accessible via docker run (image, volumes, networking, env vars...)
- Allows to define CPU and memory limits for the tasks
- Can assign an IAM role to a task
- Configurable using JSON



Service

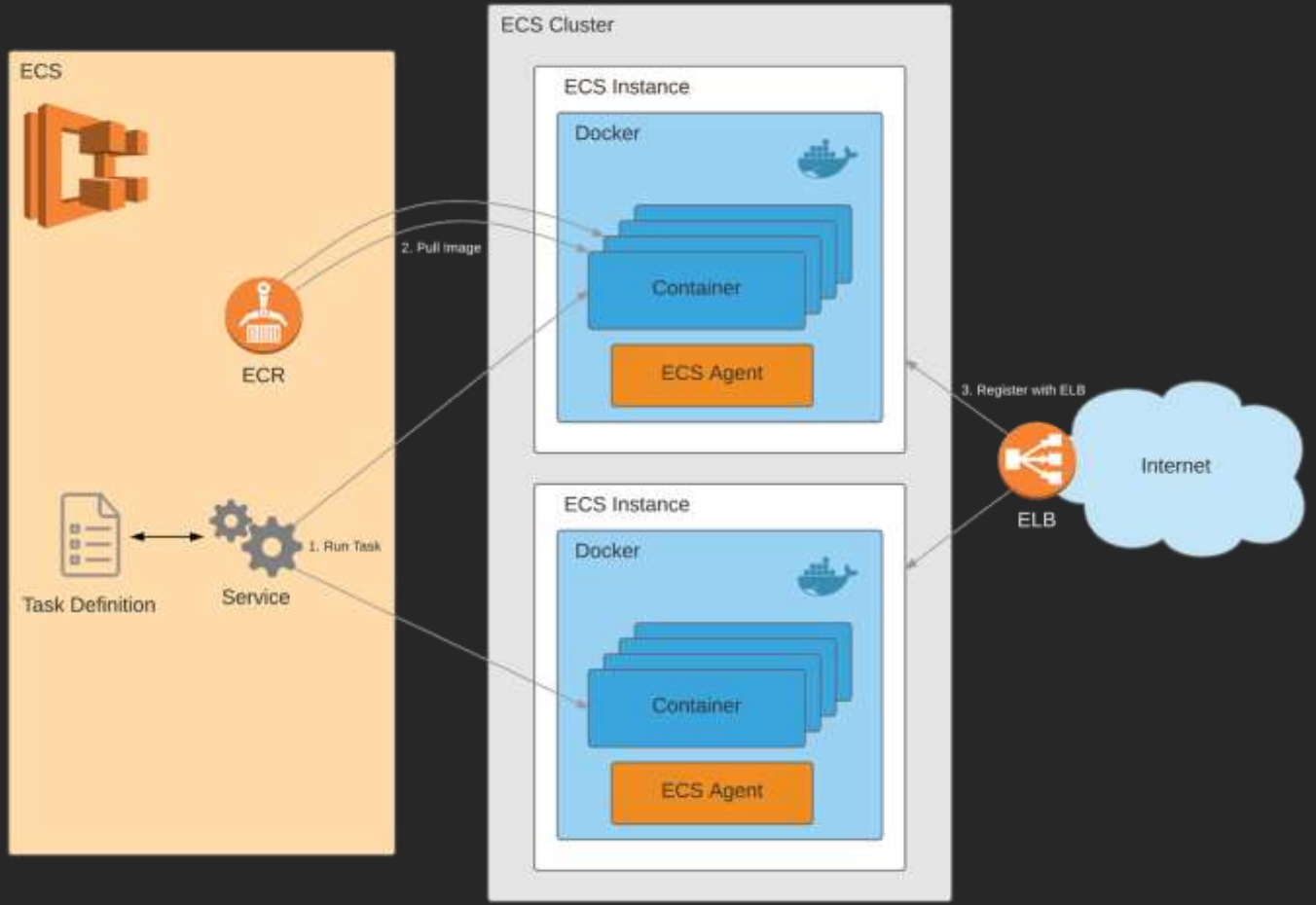
- An abstraction above tasks
- Deploys multiple “copies” from a task definition
- Maintaines the desired number of running tasks
- May bind to a load balancer



ECS Components

- **Cluster** - a group of container instances
- **Container Instance** - an EC2 instance that hosts containers
- **Task** - a set of related Docker containers
- **Task Definition** - a template which defines a task
- **Service** - a group of identical tasks

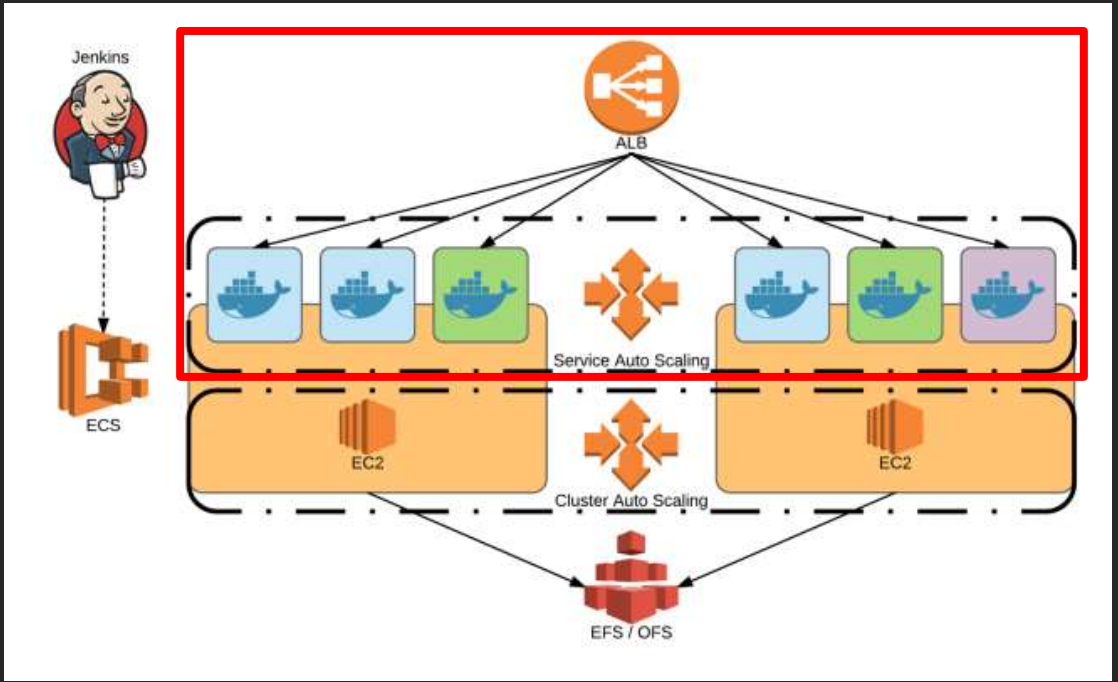
emind
YOUR CLOUD EXPERTS



ECS - Summary & Best Practices

- Use ECS to easily manage containerized apps on AWS
- Deploy ECS instances in multiple AZs for high availability
- Choose an instance type that is appropriate for your apps

Service Discovery



Question:

How does a client know where to send a request when a service runs on multiple nodes?

What is Service Discovery?

- A mechanism which allows a client to find out the network location of a service automatically

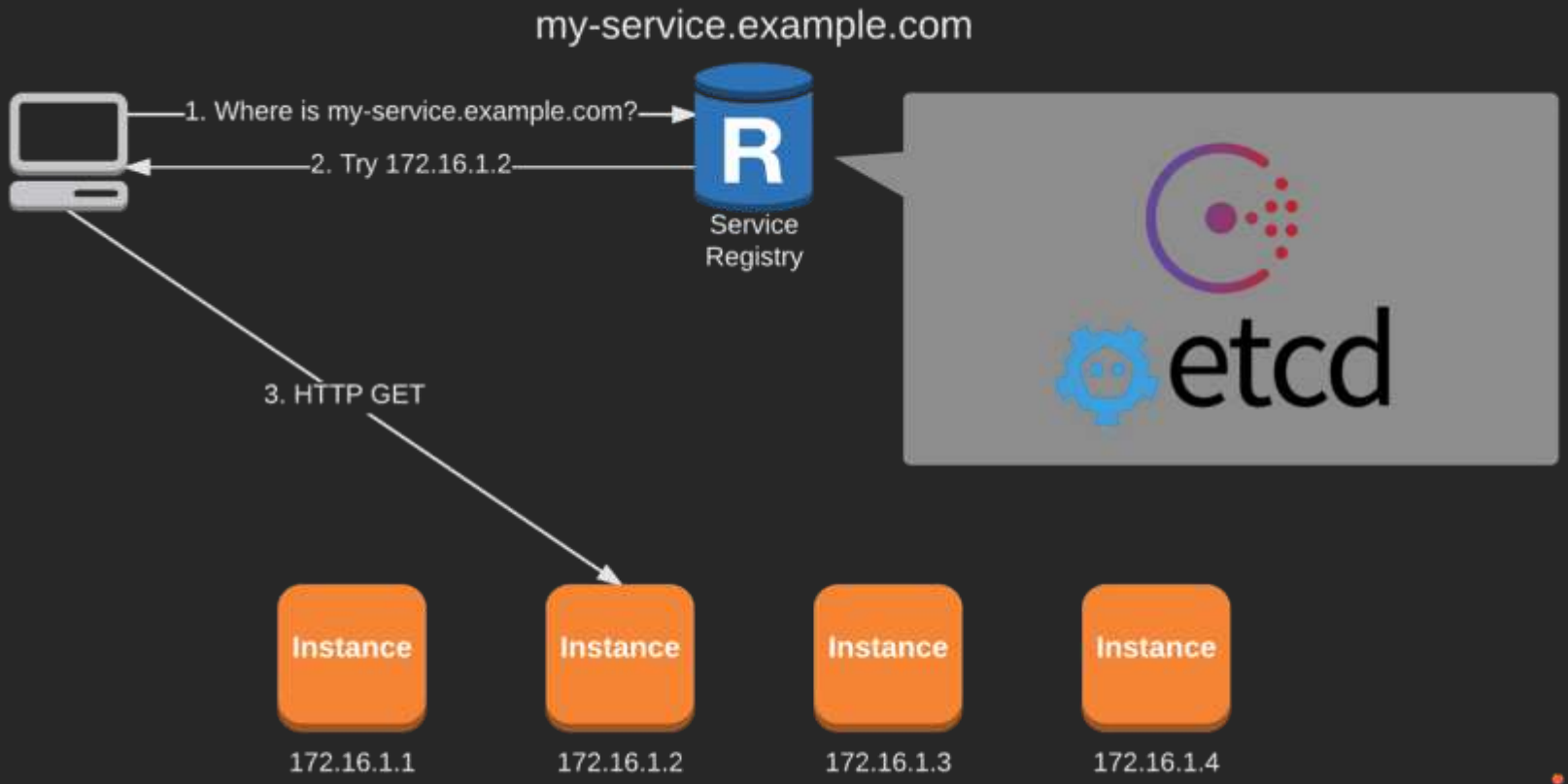
Why Do We Need Service Discovery?

- Cloud environments change all the time
- IP addresses and ports are assigned dynamically
- Auto Scaling launches and terminates instances
- Some instances might be under maintenance or upgrade

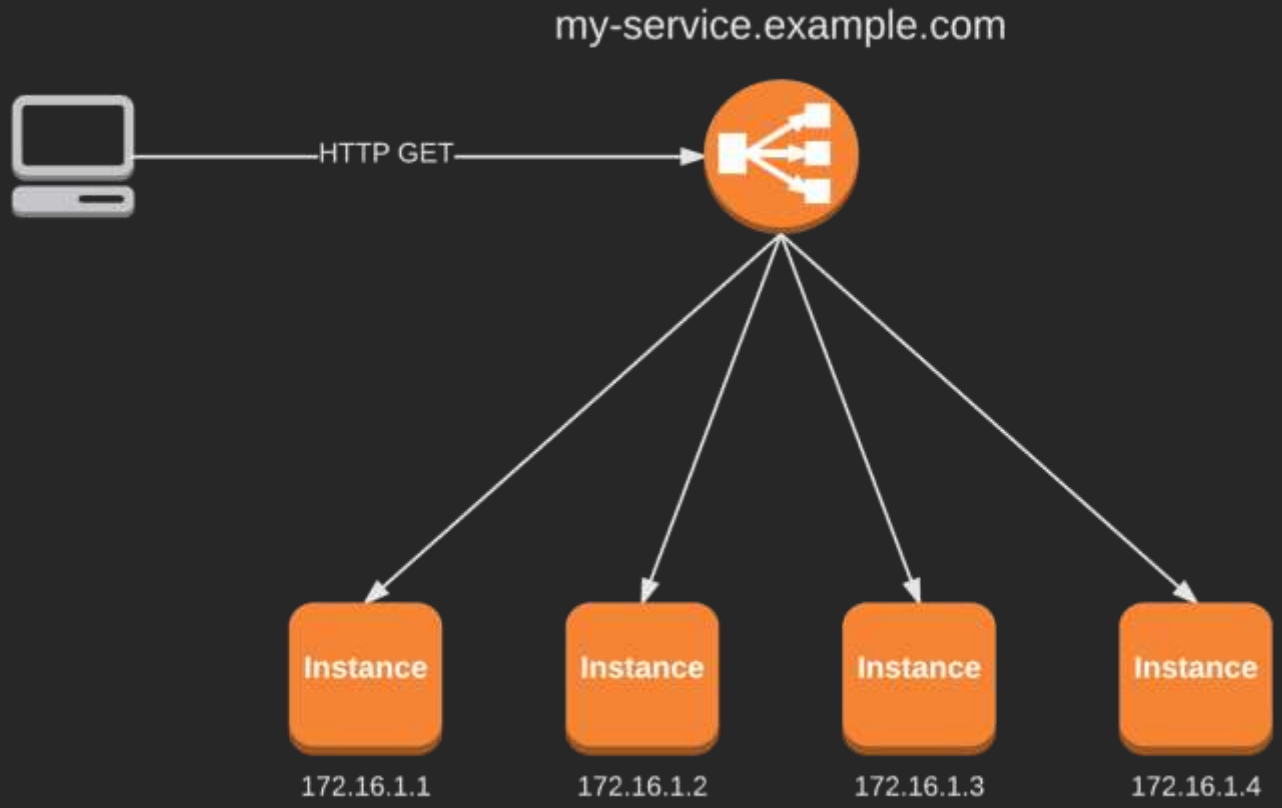
Understanding the Problem

?

Service Discovery Using a Service Registry



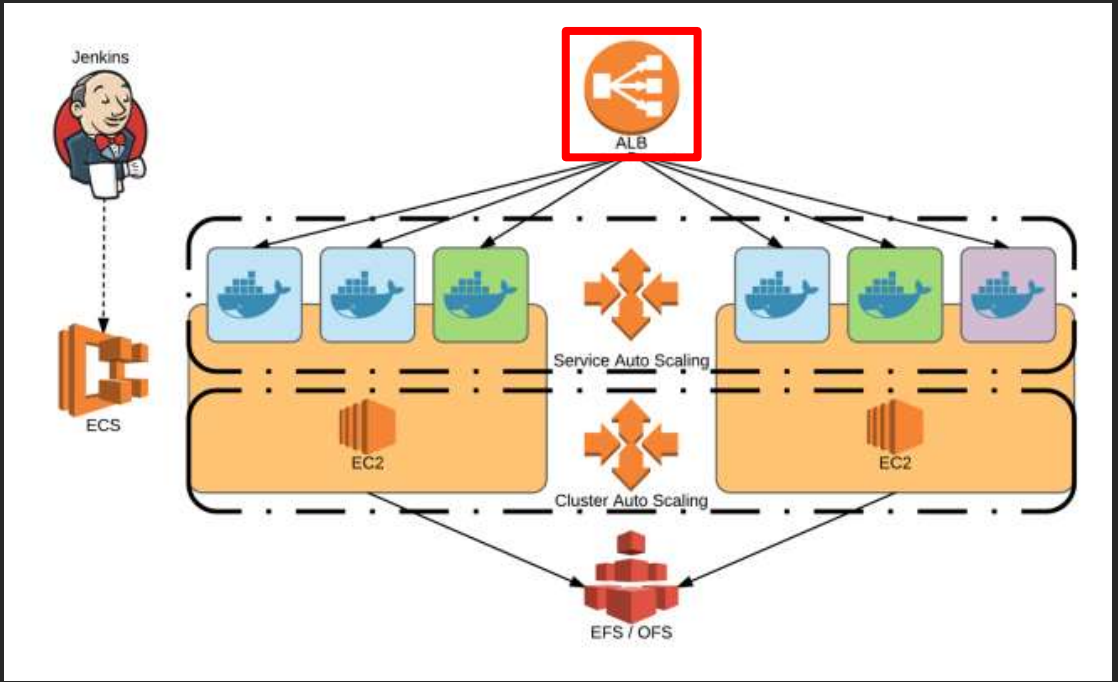
Service Discovery Using a Load Balancer



Service Discovery - Summary & Best Practices

- Cloud environments are dynamic and require service discovery
- There are multiple solutions for service discovery
- Use load balancers when possible
- Architectures combining a service registry and load balancers are possible but are more complicated

Service Load Balancing



Question:

How can we provide a single point of access to a service which runs on multiple containers?

What is Service Load Balancing?

- A mechanism which provides a single point of access to an ECS service
- Routes traffic to multiple containers
- Can be internet-facing or internal
- Powered by AWS ELB
- Complements Auto Scaling

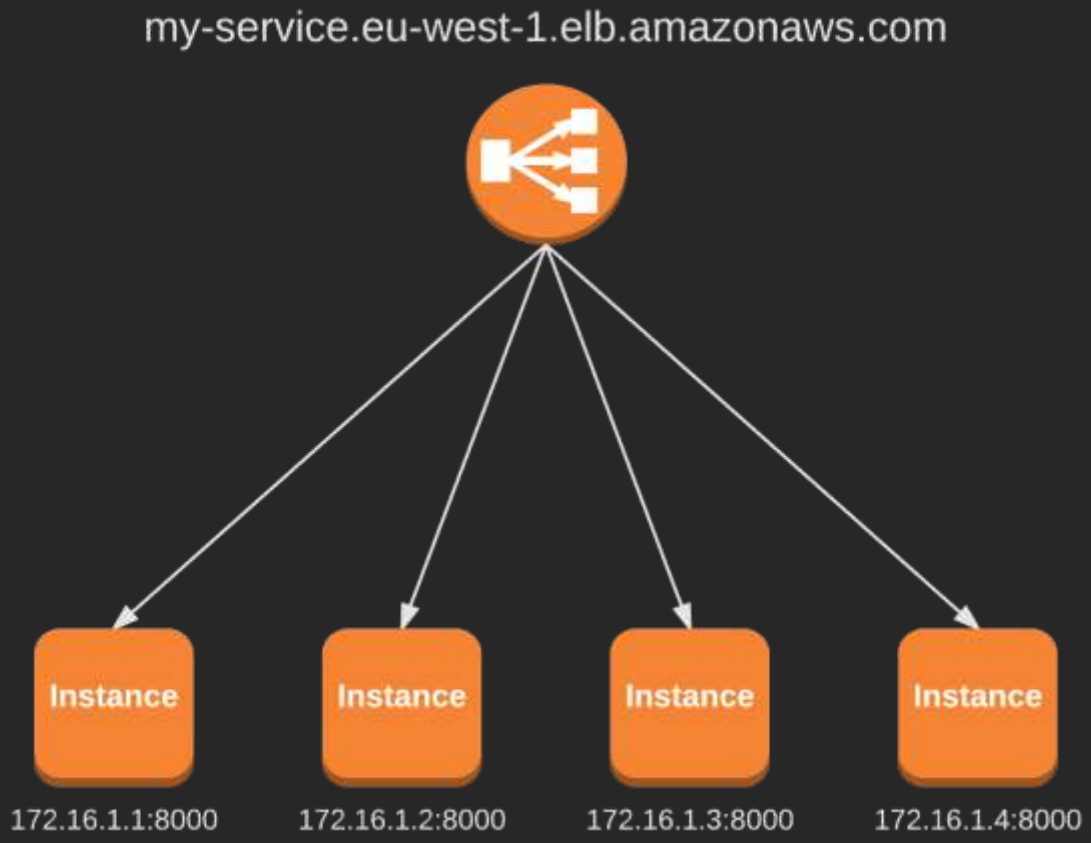
Why Use Service Load Balancing?

- Native integration with ECS
- Highly-available and auto-scaling by design
- Provides session stickiness
- Built-in health checks per service
- Support for VPC Security Groups

ELB - Classic Load Balancer

- A mature AWS service
- Routes traffic among EC2 instances
- Supports Layer 4 routing or (limited) Layer 7 routing
- No support for dynamic ports

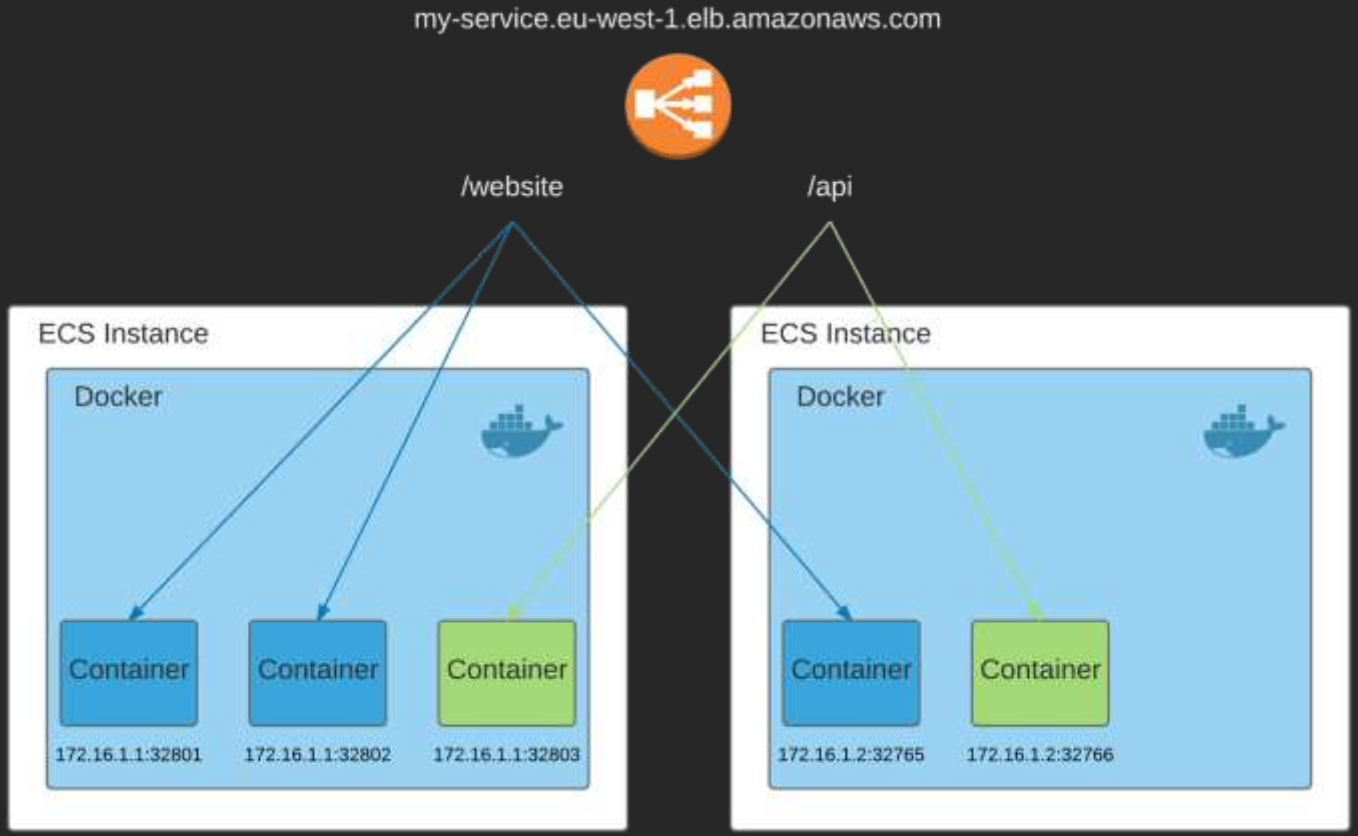
ELB - Classic Load Balancer



ELB - Application Load Balancer

- A new AWS service (announced Aug. 11, 2016)
- Supports containerized applications
- Routes traffic among EC2 instances or ECS tasks
- Supports Layer 4 routing or HTTP path-based routing
- Supports per-service health checks
- Cheaper than the classic ELB

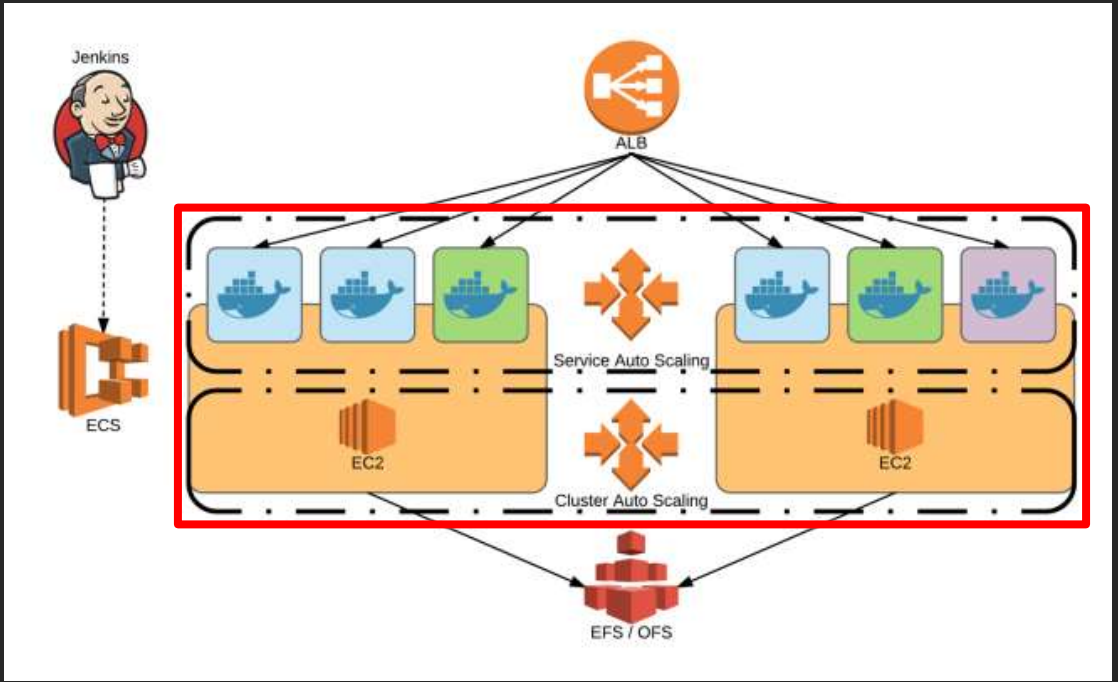
ELB - Application Load Balancer



Service Load Balancing - Summary & Best Practices

- Two types of load balancers - ELB and ALB
- Use ALBs whenever possible
- Save costs by using path-based routing - one ALB can serve a big cluster with multiple services

Auto Scaling



Question:

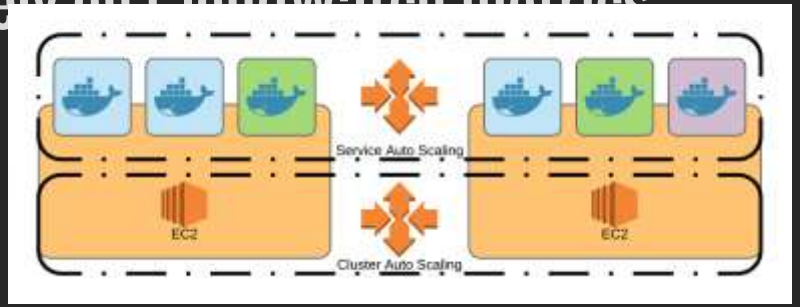
How can we automatically scale an ECS service based on load?

What is Auto Scaling?

- Automatically adjusting the capacity of the application's infrastructure based on load

Auto Scaling in ECS

- Service Auto Scaling - adjusting the number of running ECS tasks for the given service
- Cluster Auto Scaling - adjusting the number of EC2 instances in the cluster
- Both types rely on CloudWatch metrics



ECS Resource Allocation

- Each container gets a portion of the CPU and memory of the host on which it runs
- This capacity is reserved for each container
- The remaining capacity is shared among all containers
- Resource allocation is configured in the task definition

CPU Resource Allocation

- Each ECS instance has 1024 CPU units per CPU core
- A container gets a relative amount of CPU cycles based on the configured units
- The configured units are reserved for the container
- CPU allocation is only relevant when there is competition on host resources
- The remaining CPU capacity may be used by other containers

Memory Resource Allocation

- Soft limit - the amount is reserved for the container but may be exceeded if capacity is available
- Hard limit - container is killed when trying to exceed the reserved amount
- Must use one limit type but may use both together

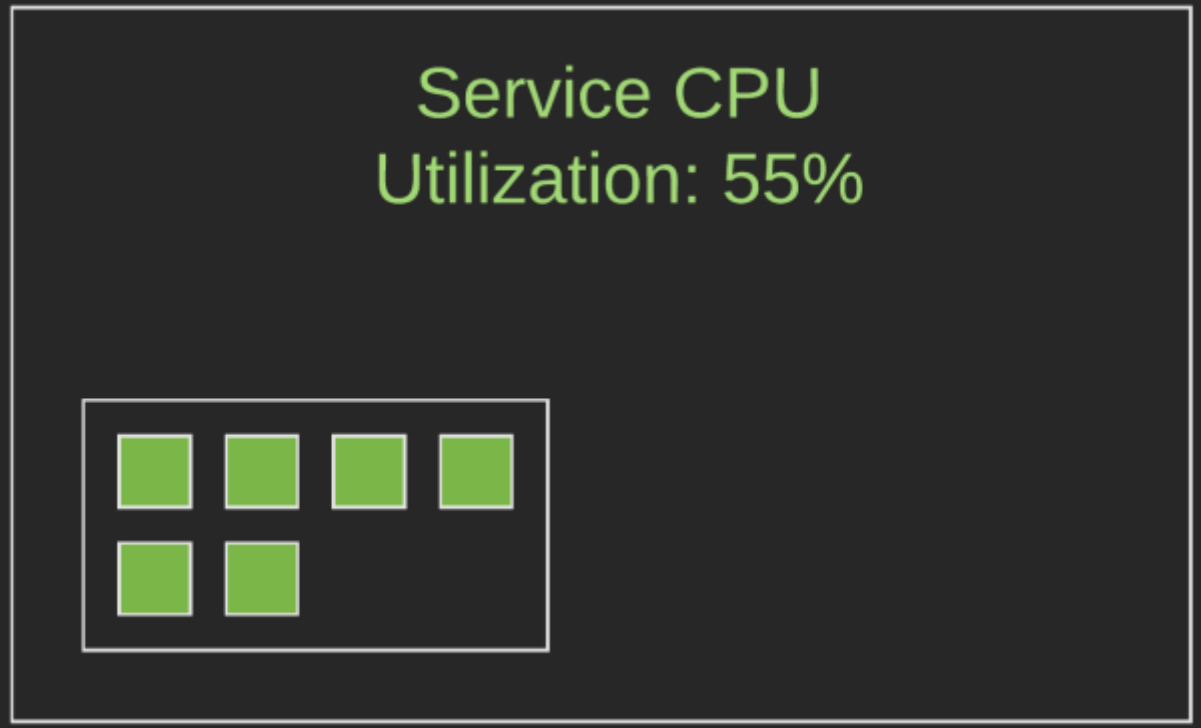
Service Auto Scaling

- Adding more containers to handle an increasing load
- Configured inside ECS
- Use CPU and memory usage to trigger scaling events
- May use custom CloudWatch metrics too
- “Do we have enough compute power?”

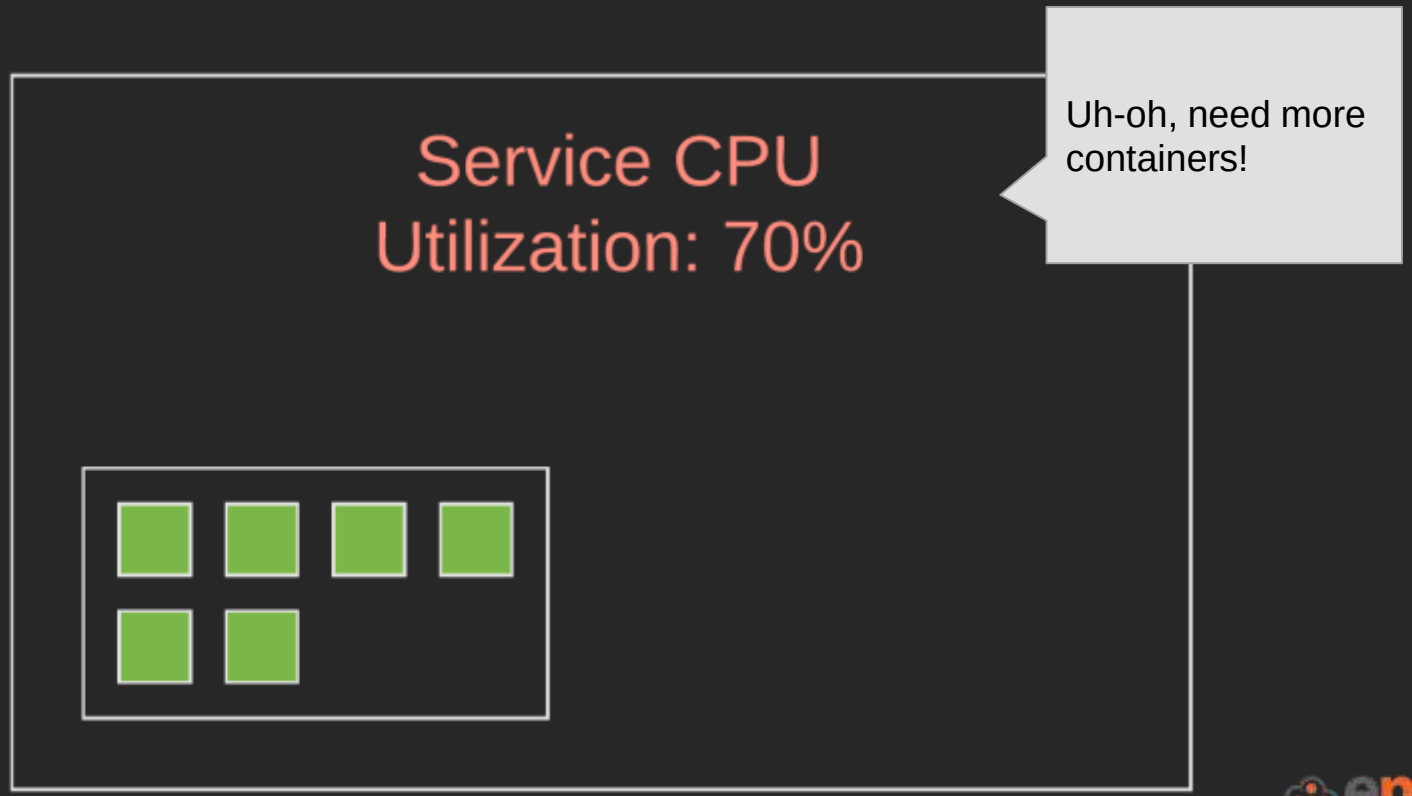
Cluster Auto Scaling

- Adding more instances to accommodate an increasing number of containers
- Configured via EC2 Auto Scaling
- Use CPU and memory reservation to trigger scaling events
- “Do we have room for more containers?”

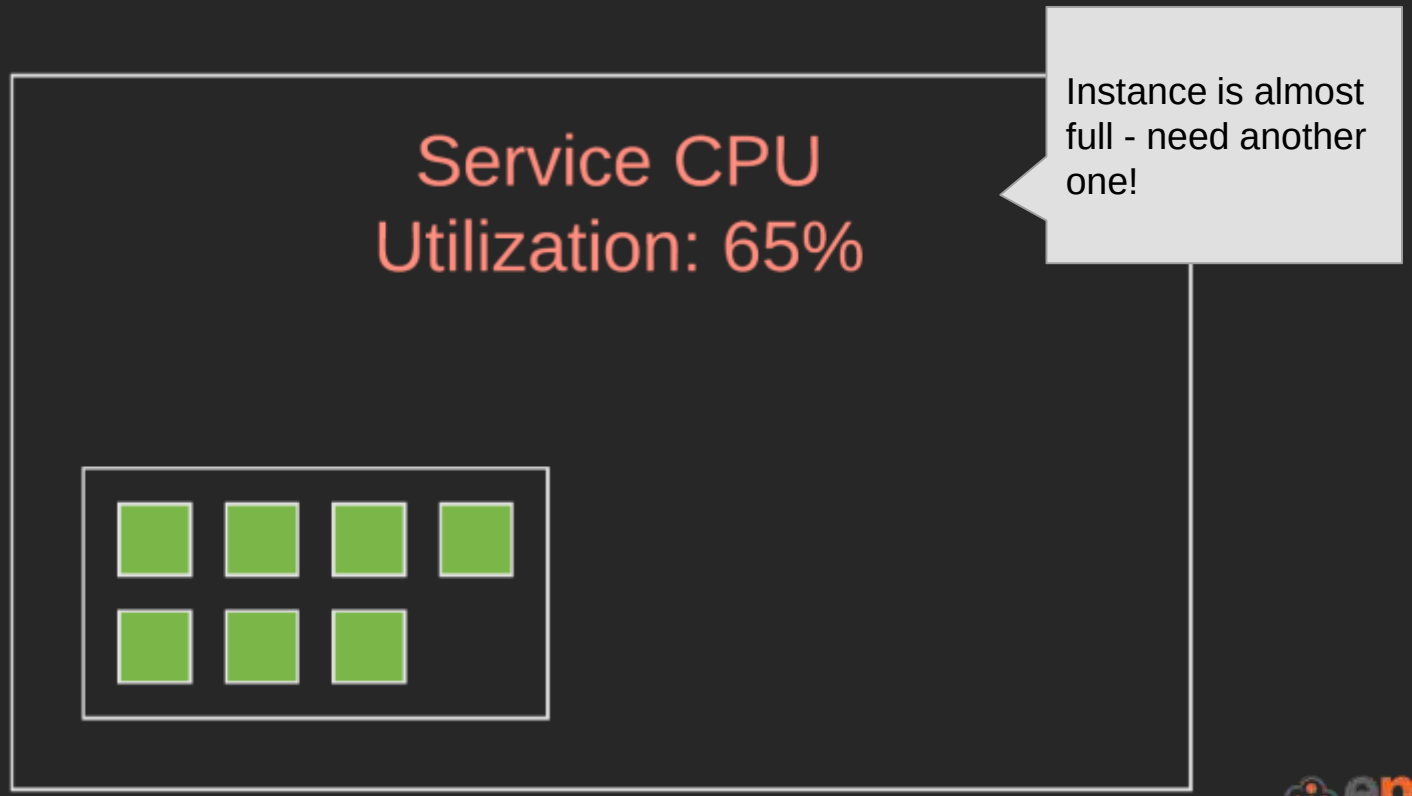
Auto Scaling in Action



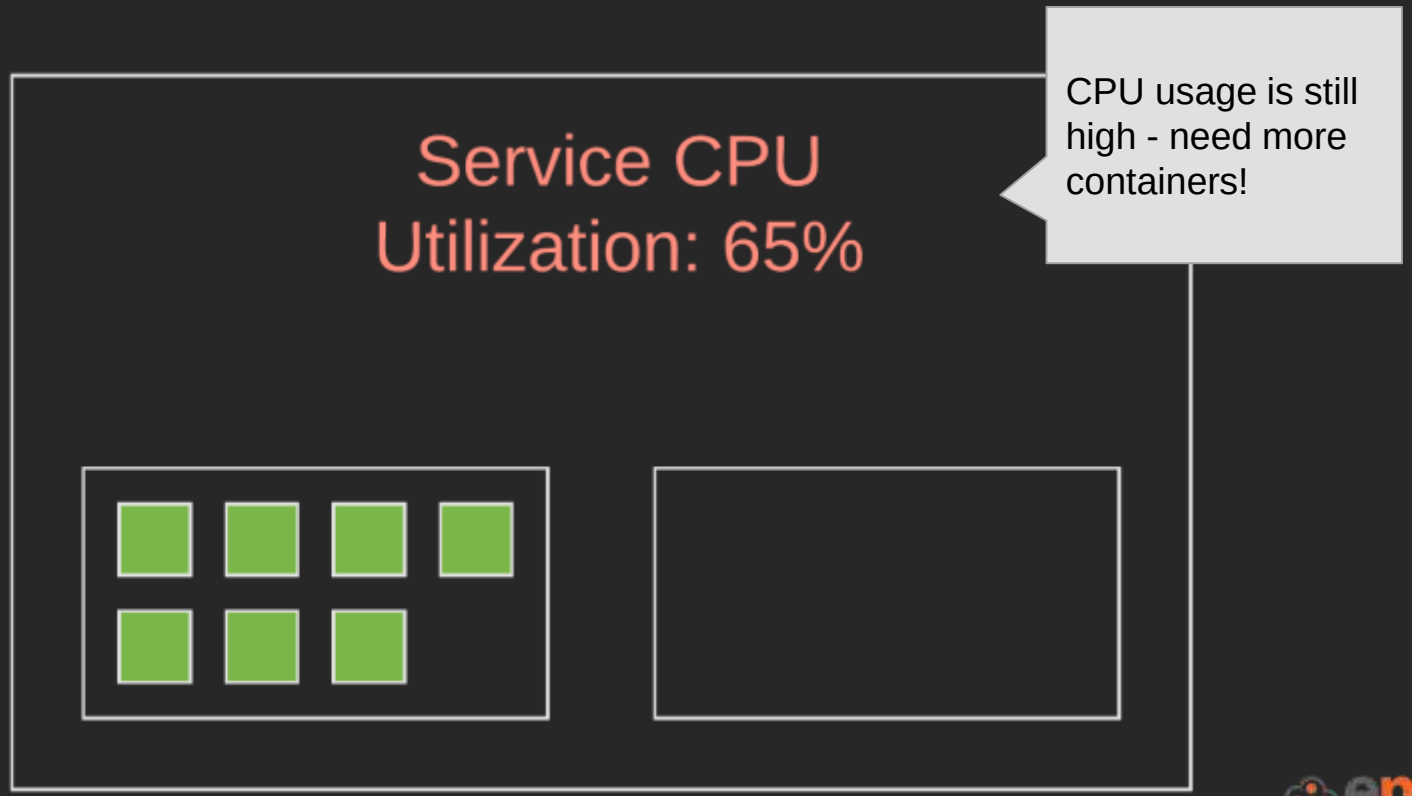
Auto Scaling in Action



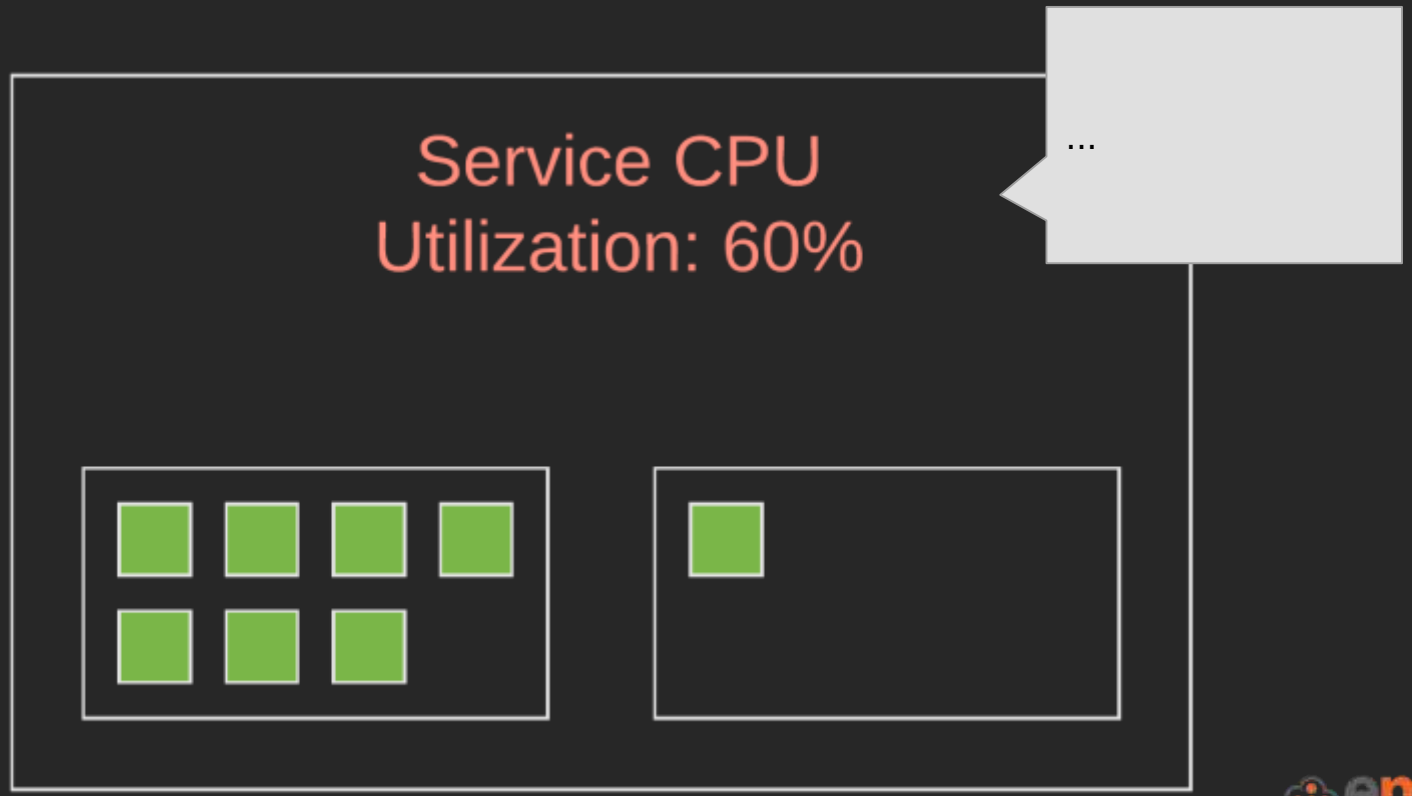
Auto Scaling in Action



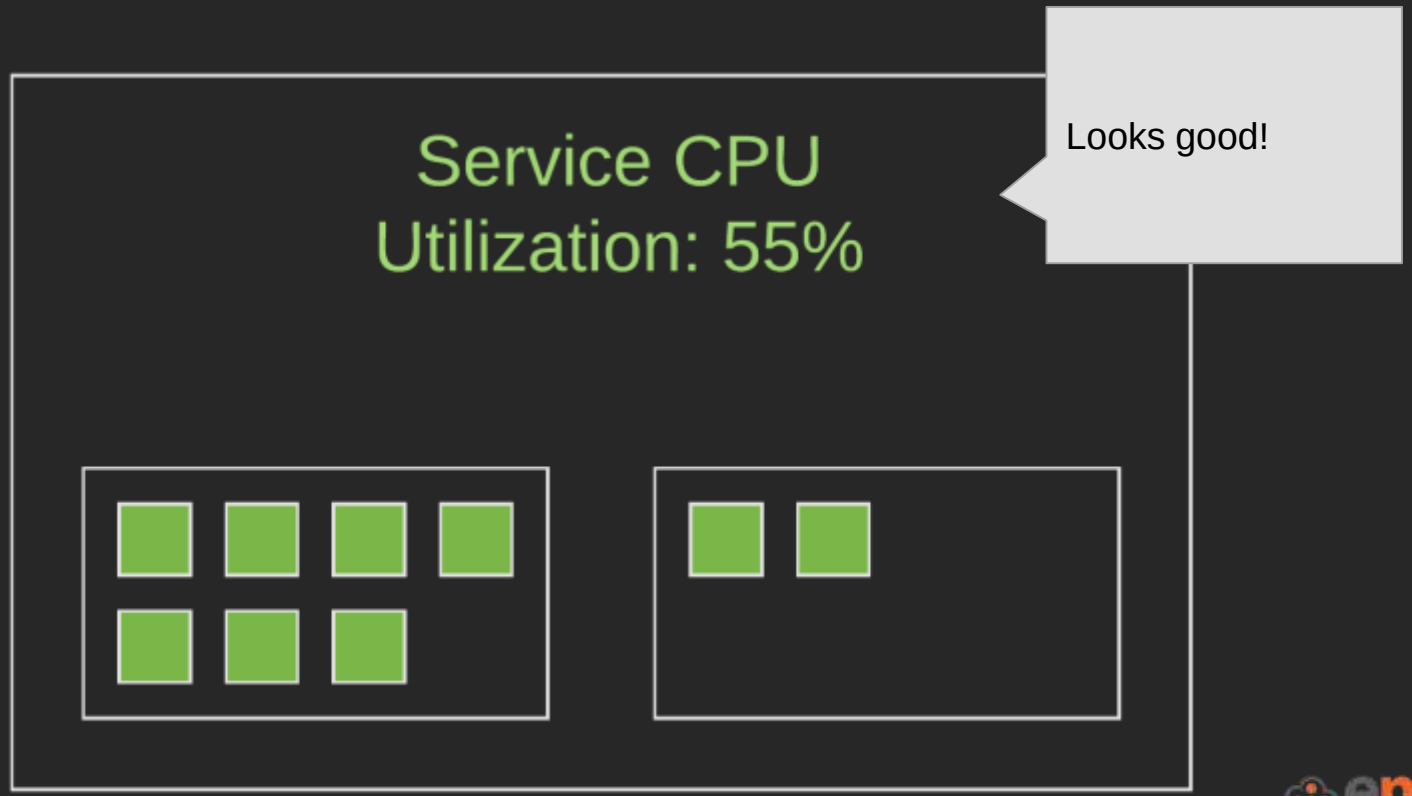
Auto Scaling in Action



Auto Scaling in Action



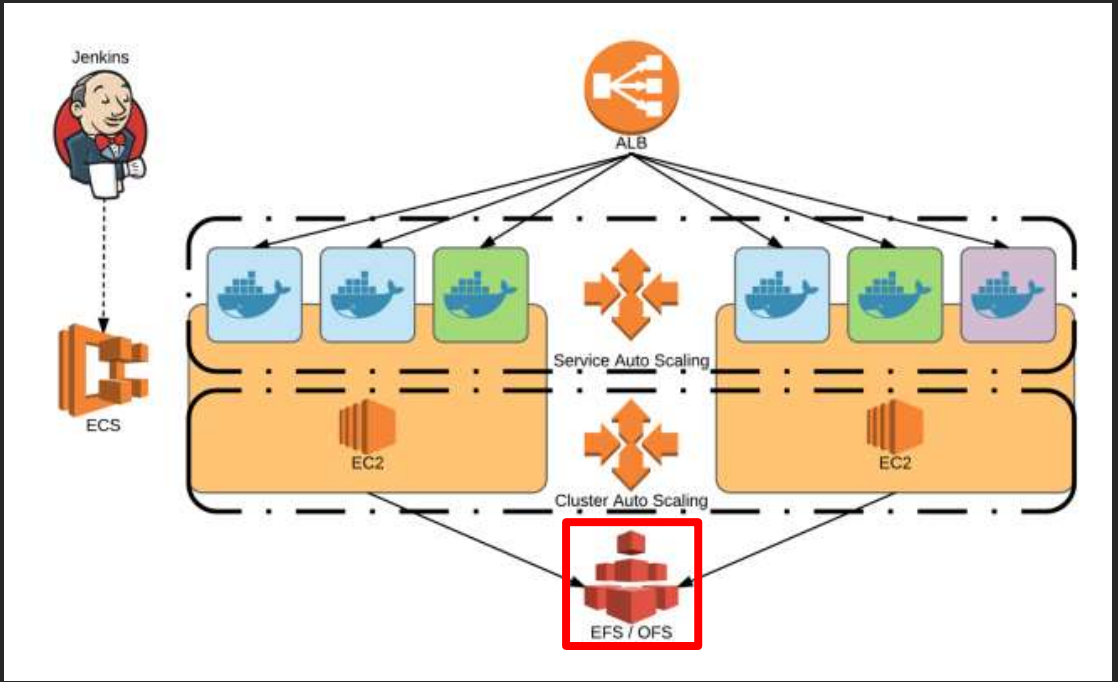
Auto Scaling in Action



Auto Scaling - Summary & Best Practices

- Configure both Service Auto Scaling and Cluster Auto Scaling
- Scale services based on utilization
- Scale clusters based on reservation
- Service Auto Scaling is much faster than Cluster Auto Scaling
- Leave some spare capacity on each host
  - Allows the cluster to scale in time

Storage



Question:

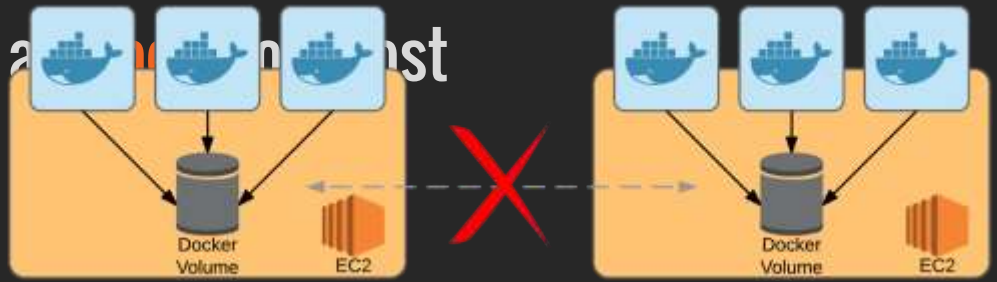
How to persist data used by a containerized application and share it among containers on multiple hosts?

Storage in Docker

- Docker containers are volatile
- Docker uses Union File Systems for container storage
- Data that is written to the Union File System doesn't persist

Docker Volumes

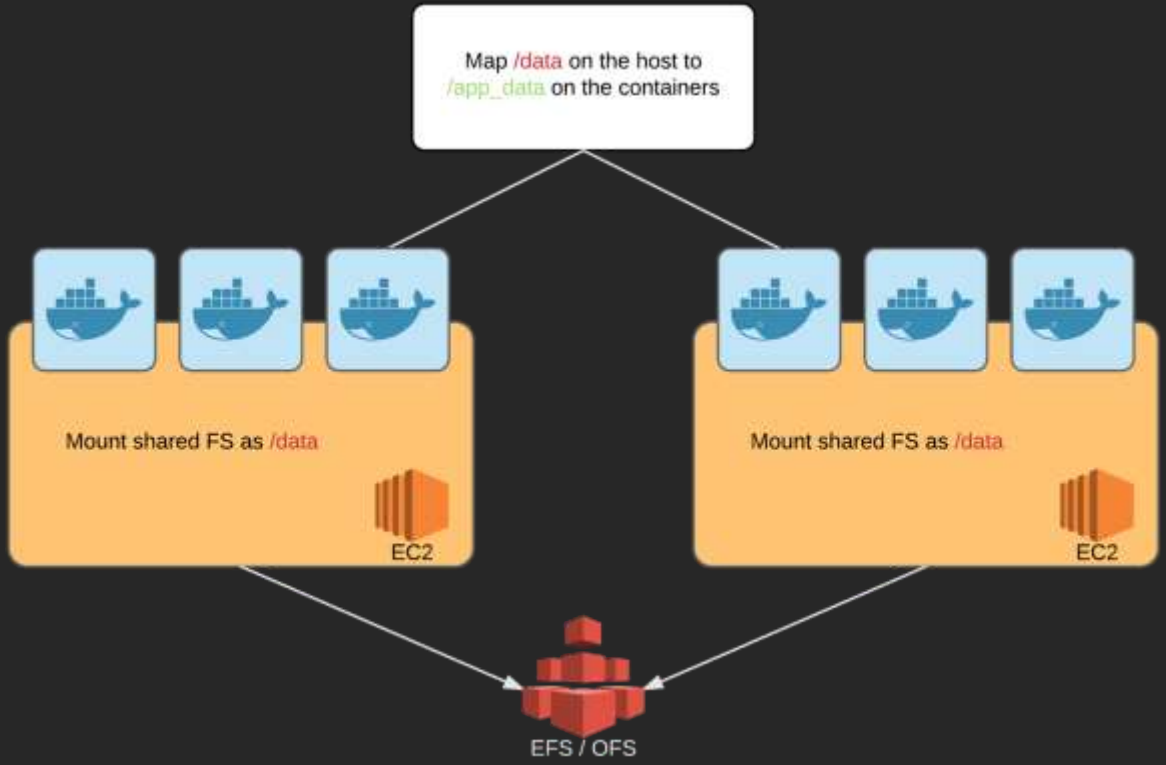
- Docker volumes can be used to persist data and share data between containers
- Docker volumes bypass the Union File System
- Host directories may be mounted as volumes
- Volumes a[illegible]st



Shared File Systems

- Elastic File System (EFS) - a shared storage solution by AWS
- ObjectiveFS - a 3rd party shared storage solution on top of S3
- Both solutions provide the following:
  - A shared file system which can be accessed by multiple servers at the same time
  - Unlimited capacity which expands automatically
  - High availability by design

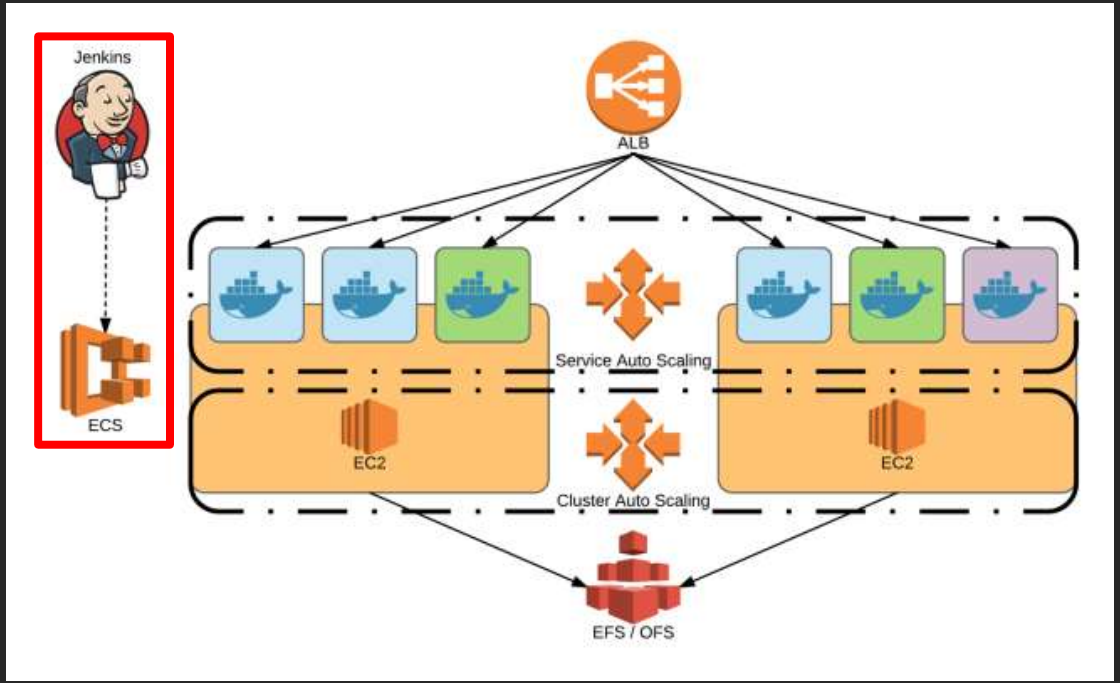
Using a Shared File System



Storage - Summary & Best Practices

- Use Docker volumes for persistence and for sharing data between containers
- Mount a shared file system on each host and map Docker volumes to it

Continuous Integration & Delivery



Question:

How to deploy applications to ECS and update them without service disruption?

CI/CD with ECS

- ECS can use Docker images from ECR or any other registry
- You can specify which images to deploy using task definitions
- ECS allows you to perform rolling updates to running services
- Updates can be triggered automatically using the ECS API
- Jenkins or any other CI/CD solution may be used to automate the process

CI/CD with ECS - Workflow

1. Checkout source from version control to Jenkins server
2. Build a new Docker image
3. Push the new image to ECR
4. Update the task definition & service
5. ECS updates the containers on the cluster

emind
YOUR CLOUD EXPERTS

Using Docker Tags

- Docker tags allow you to manage Docker images easily
- When building a new Docker image you must tag it
- Any string may be used as a tag
- The "latest" tag is used as a default tag if no tag is specified when building an image or running a container

The “latest” Tag is Dangerous!

- Using the “latest” tag in CI/CD may lead to problems
- Pushing an image with a tag that already exists in the repository will cause that tag to move to the new image
- This can lead to two containers which appear to use the same image but in fact have different code
- A good use for “latest” is to indicate a stable or default version on a public Docker repository

Tagging Strategy

- It is important to implement a proper tagging strategy when using Docker for CI/CD
- Common tag values:
  - Application version (“1.3”)
  - CI/CD build number (“136”)
  - Git SHA value (“ca82a6d”)
- Using one of the above will make sure your tags are unique

CI/CD - Summary & Best Practices

- Use Jenkins to build new Docker images and push them to ECR
- Use Jenkins to trigger rolling updates on ECS
- Implement a proper tagging strategy
- Use the “latest” carefully and in addition to a version tag

Thank You!

johananl@emind.co
info@emind.co
jobs@emind.co

emind
YOUR CLOUD EXPERTS

Open Positions

DevOps Engineers
Cloud Architect
Big Data Specialist

emind
YOUR CLOUD EXPERTS