Introduction to Docker

Agenda

Section 1:

What is Docker

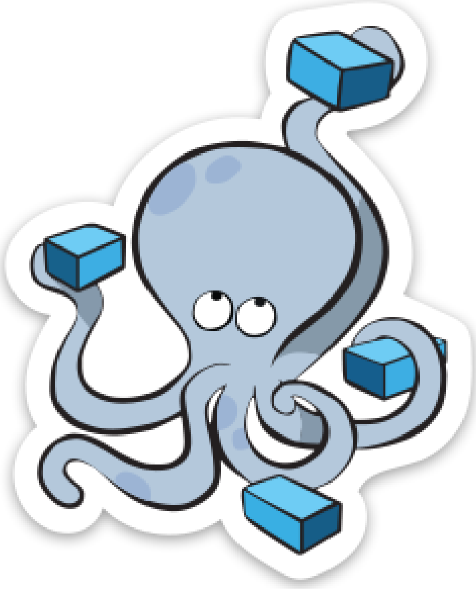
What is Docker Not

Basic Docker Commands
Dockerfiles

Section 2:

Anatomy of a Docker image

Docker volumes



Section 3:

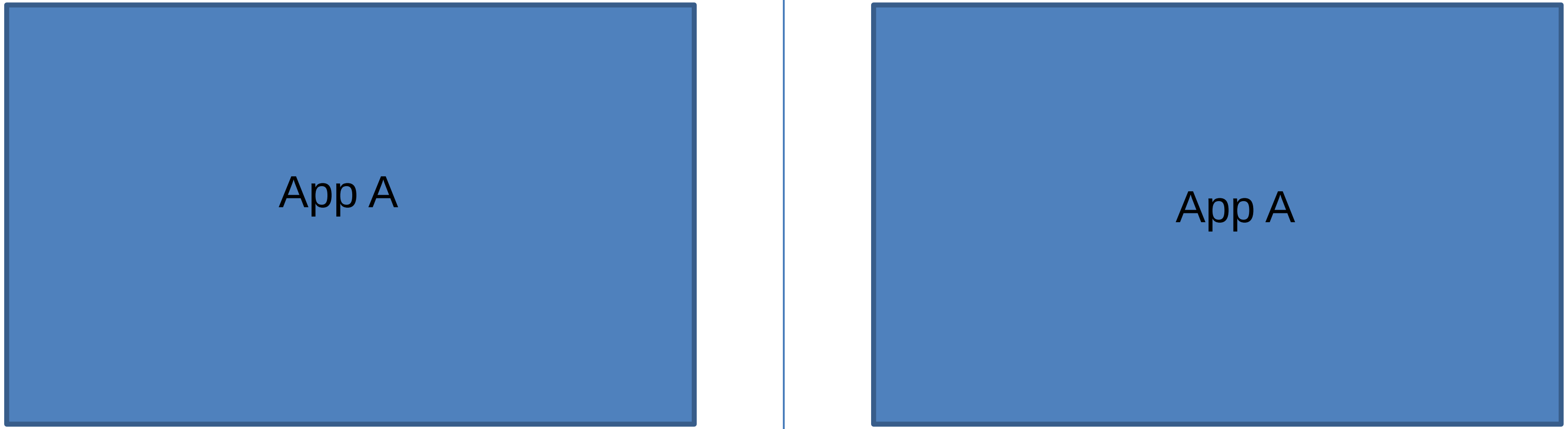
Networking

Section 4:

Docker compose / stacks

Demo

FIRST OF ALL!

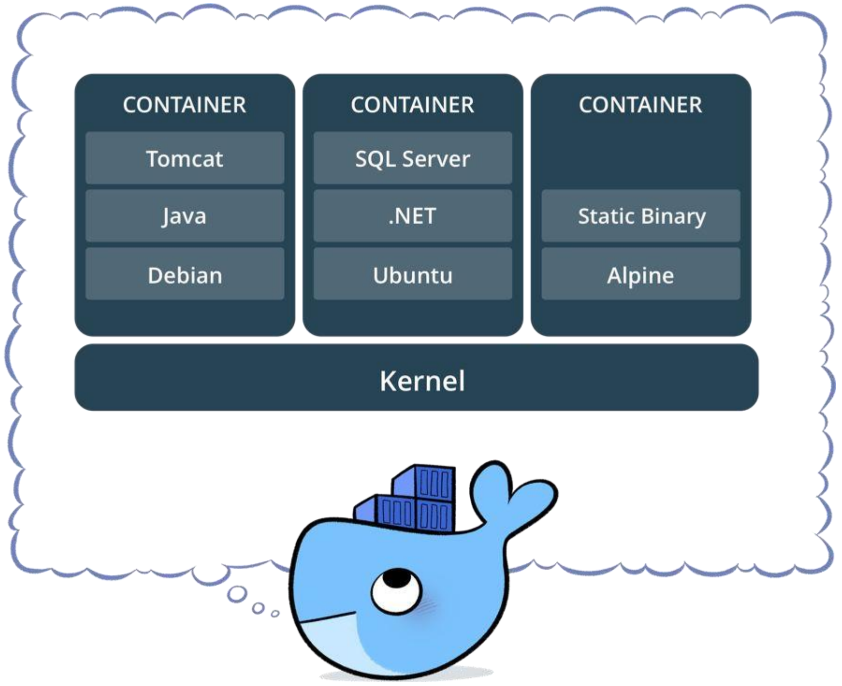


Maquina programador/Entorno desarrollo

Servidor/Entorno producción

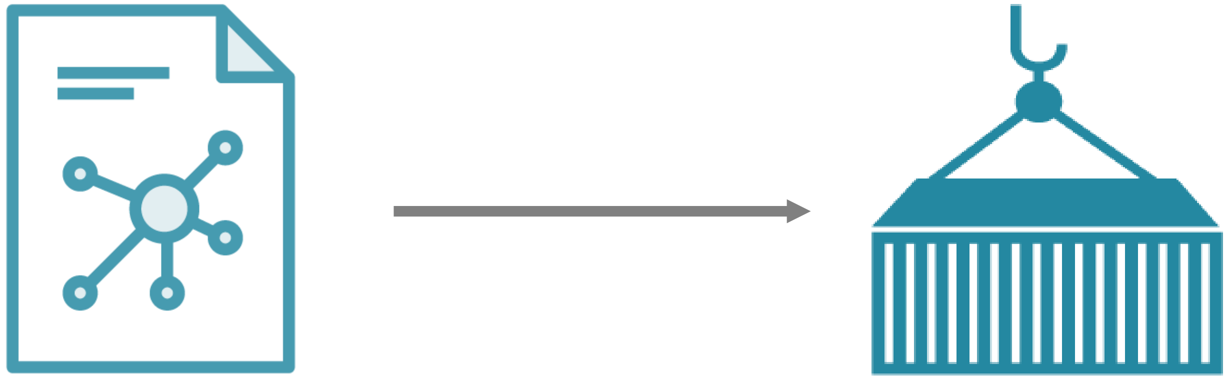
Section 1:

What is a container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works for all major Linux distributions
- Containers native to Windows Server 2016

The Role of Images and Containers



Docker Image

Example: Ubuntu with Node.js and Application Code

Docker Container

Created by using an image. Runs your application.

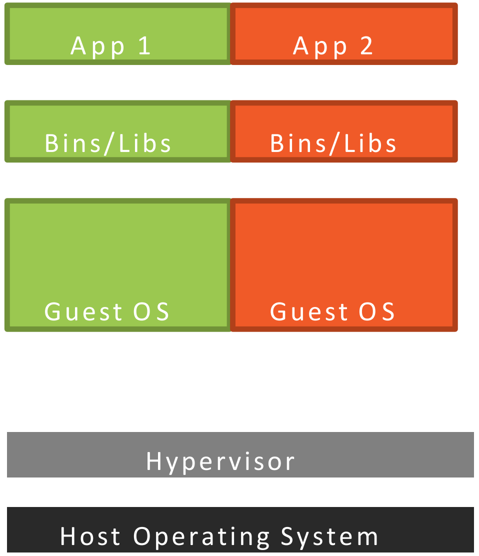
Docker containers are NOT VMs

- Easy connection to make
- Fundamentally different architectures
- Fundamentally different benefits

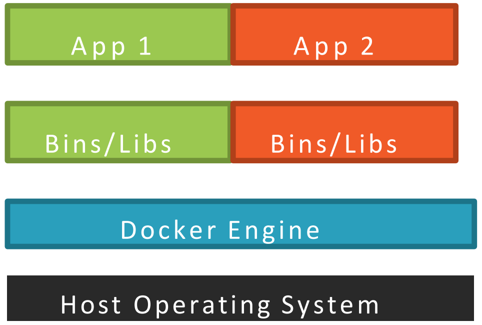
Maquina Virtual

Contenedores

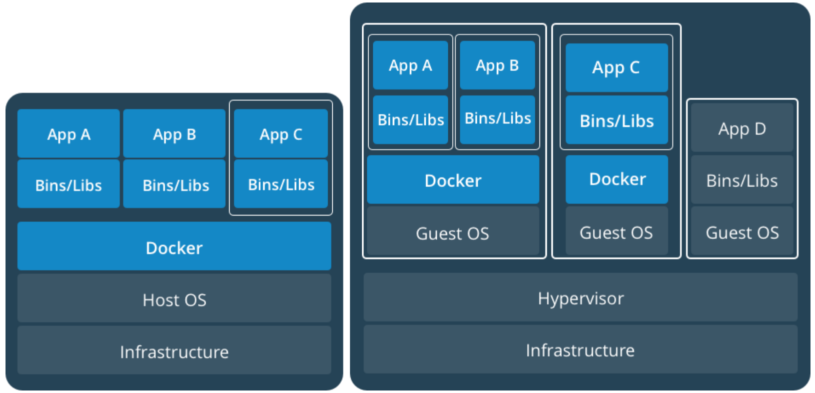
Docker Containers Versus Virtual Machines



Virtual Machines



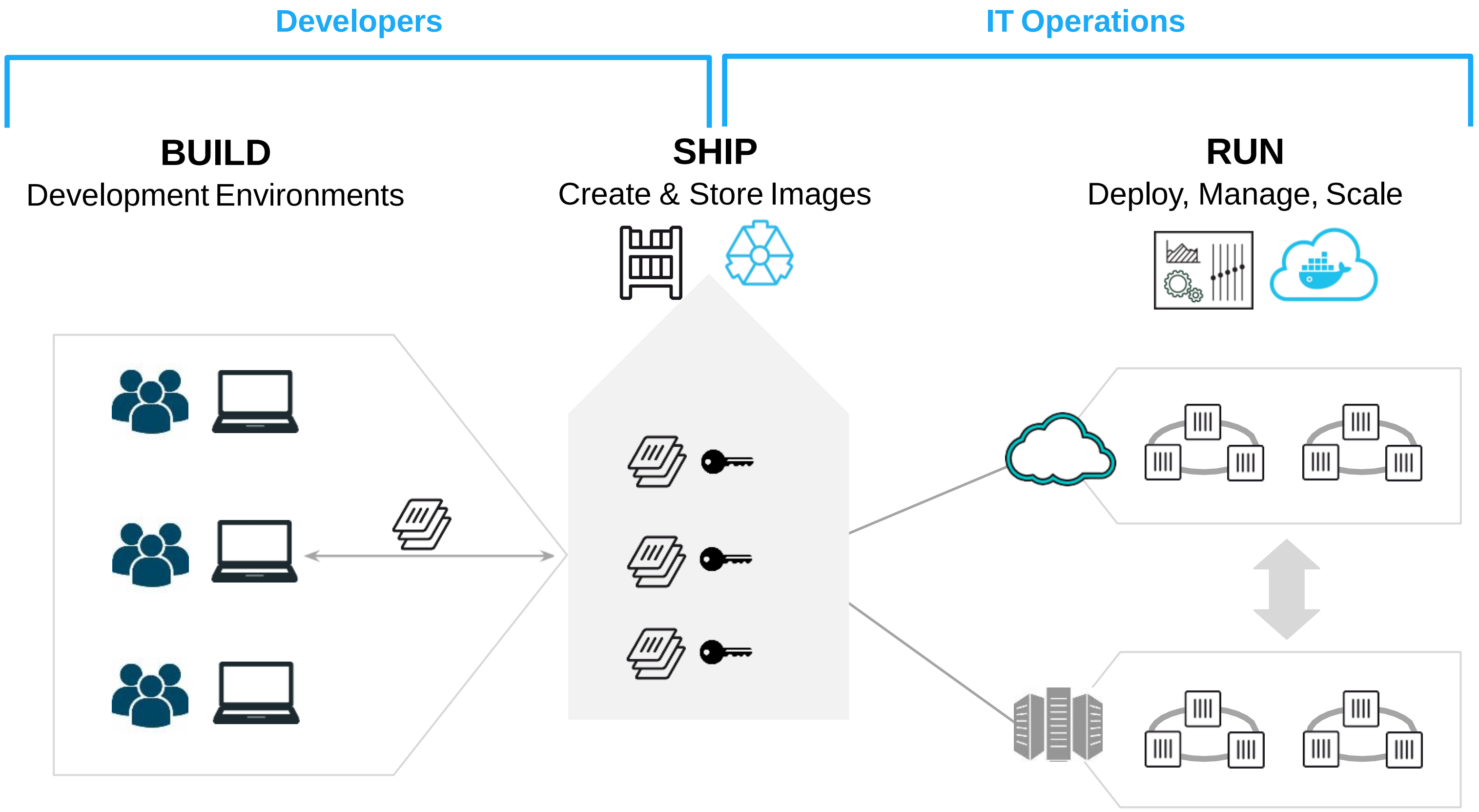
Docker Containers



What Is Docker?

- Lightweight, open, secure platform
- Simplify building, shipping, running apps
- Runs natively on Linux or Windows Server
- Runs on Windows or Mac Development machines (with a virtual machine)
- Relies on "images" and "containers"

Using Docker: Build, Ship, Run Workflow



Some Docker vocabulary

Docker Image

The basis of a Docker container. Represents a full application

Docker Container

The standard unit in which the application service resides and executes

Docker Engine

Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

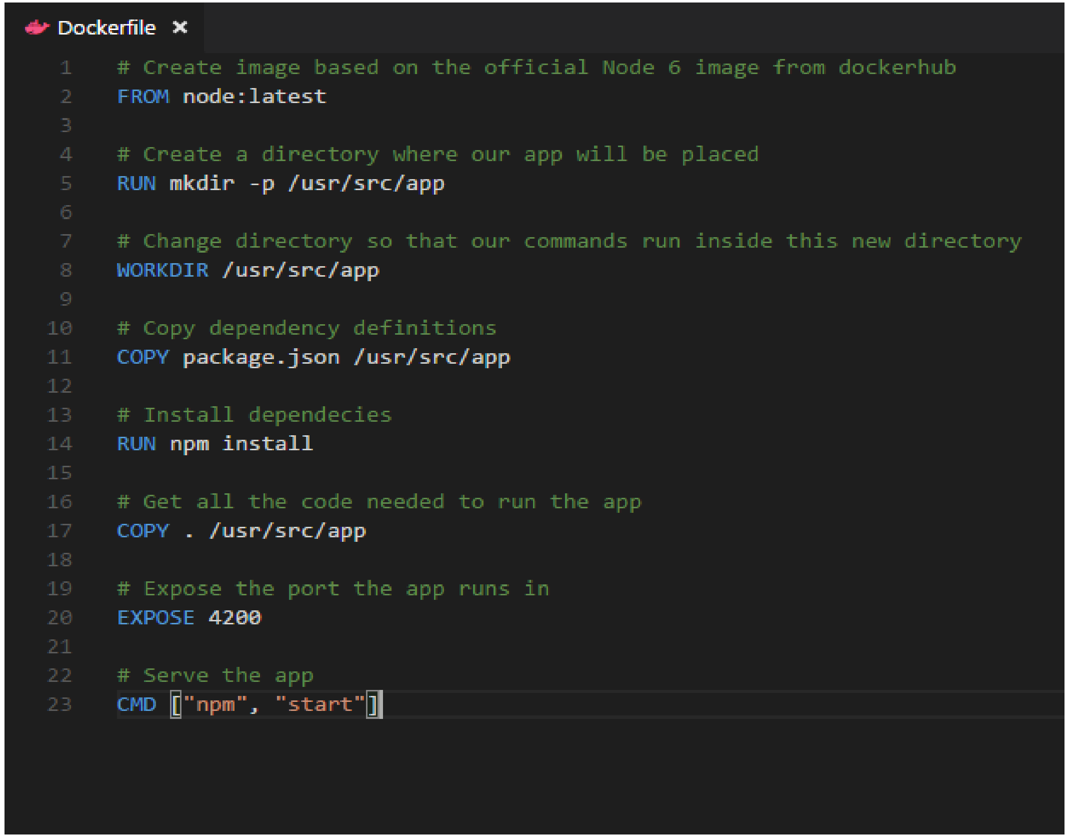
Registry Service (Docker Hub(Public) or Docker Trusted Registry(Private))

Cloud or server based storage and distribution service for your images

Basic Docker Commands

```
$ docker image pull node:latest

$ docker image ls
$ docker container run -d -p 5000:5000 --name node node:latest

$ docker container ps


$ docker container stop node(or <container id>)

$ docker container rm node (or <container id>)

$ docker image rmi (or <image id>)

$ docker build -t node:2.0 .

$ docker image push node:2.0

$ docker --help
```

Dockerfile – Linux Example



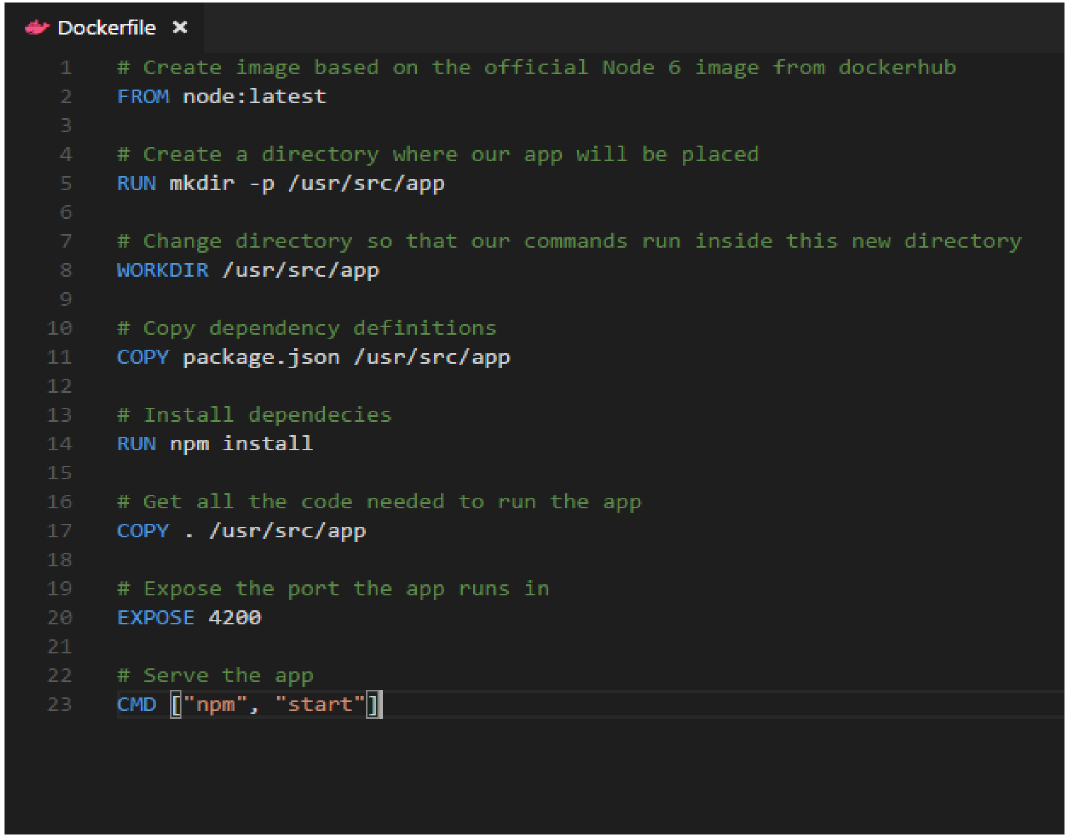
Dockerfile ×

```
# Create image based on the official Node 6 image from dockerhub
FROM node:latest

# Create a directory where our app will be placed
RUN mkdir -p /usr/src/app

# Change directory so that our commands run inside this new directory
WORKDIR /usr/src/app

# Copy dependency definitions
COPY package.json /usr/src/app

# Install dependecies
RUN npm install

# Get all the code needed to run the app
COPY . /usr/src/app

# Expose the port the app runs in
EXPOSE 4200

# Serve the app
CMD ["npm", "start"]
```

- Instructions on how to build a Docker image
- Looks very similar to “native” commands
- Important to optimize your Dockerfile

Section 2:

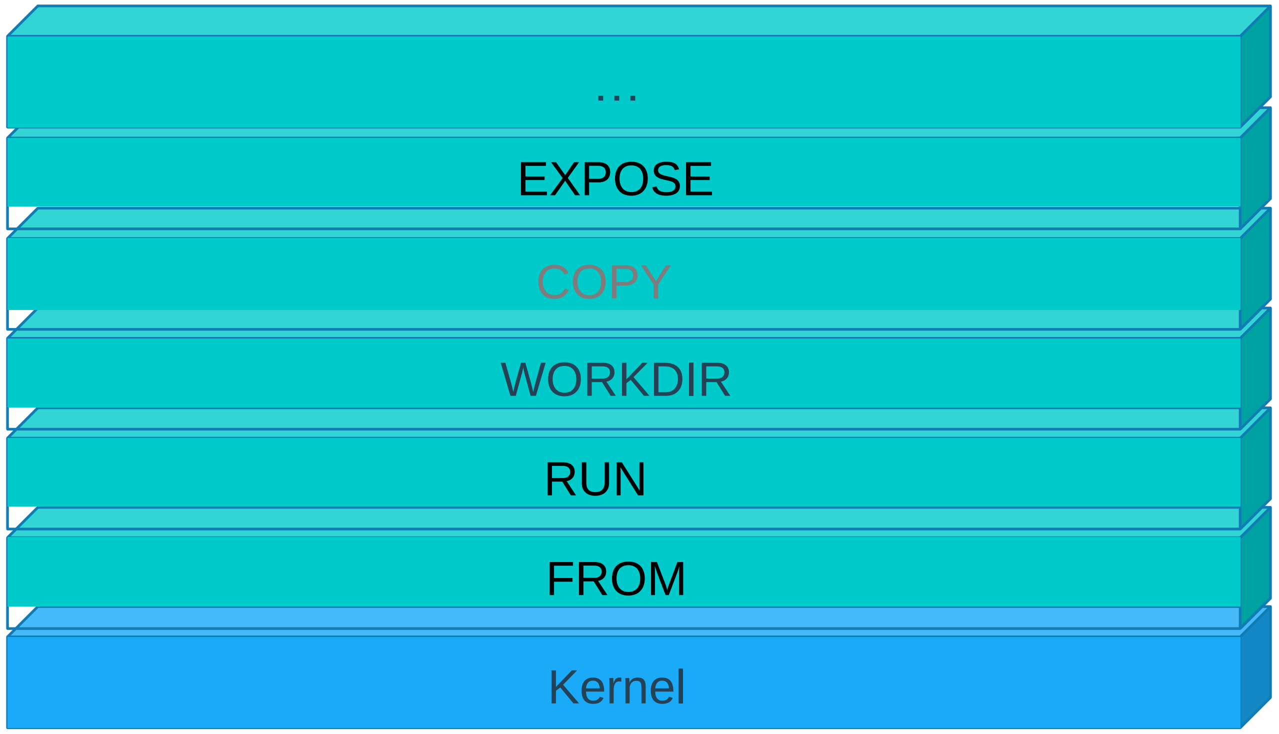
Let’s Go Back to Our Dockerfile



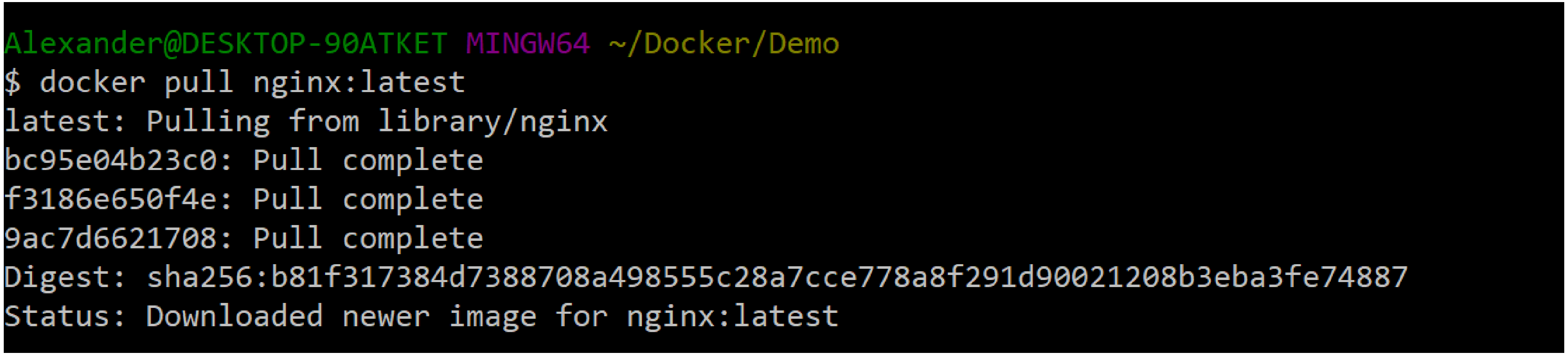
Dockerfile ×

```
# Create image based on the official Node 6 image from dockerhub
FROM node:latest

# Create a directory where our app will be placed
RUN mkdir -p /usr/src/app

# Change directory so that our commands run inside this new directory
WORKDIR /usr/src/app

# Copy dependency definitions
COPY package.json /usr/src/app

# Install dependecies
RUN npm install

# Get all the code needed to run the app
COPY . /usr/src/app

# Expose the port the app runs in
EXPOSE 4200

# Serve the app
CMD ["npm", "start"]
```

Each Dockerfile Command Creates a Layer



Docker Image Pull: Pulls Layers



```
Alexander@DESKTOP-90ATKET MINGW64 ~/Docker/Demo
$ docker pull nginx:latest
latest: Pulling from library/nginx
bc95e04b23c0: Pull complete
f3186e650f4e: Pull complete
9ac7d6621708: Pull complete
Digest: sha256:b81f317384d7388708a498555c28a7cce778a8f291d90021208b3eba3fe74887
Status: Downloaded newer image for nginx:latest
```

Docker Volumes

- Volumes mount a directory on the host into the container at a specific location

- Can be used to share (and persist) data between containers
  - Directory persists after the container is deleted
    - Unless you explicitly delete it

- Can be created in a Dockerfile or via CLI

Why Use Volumes

- Mount local source code into a running container

```
docker container run -v $(pwd):/usr/src/app/
myapp
```

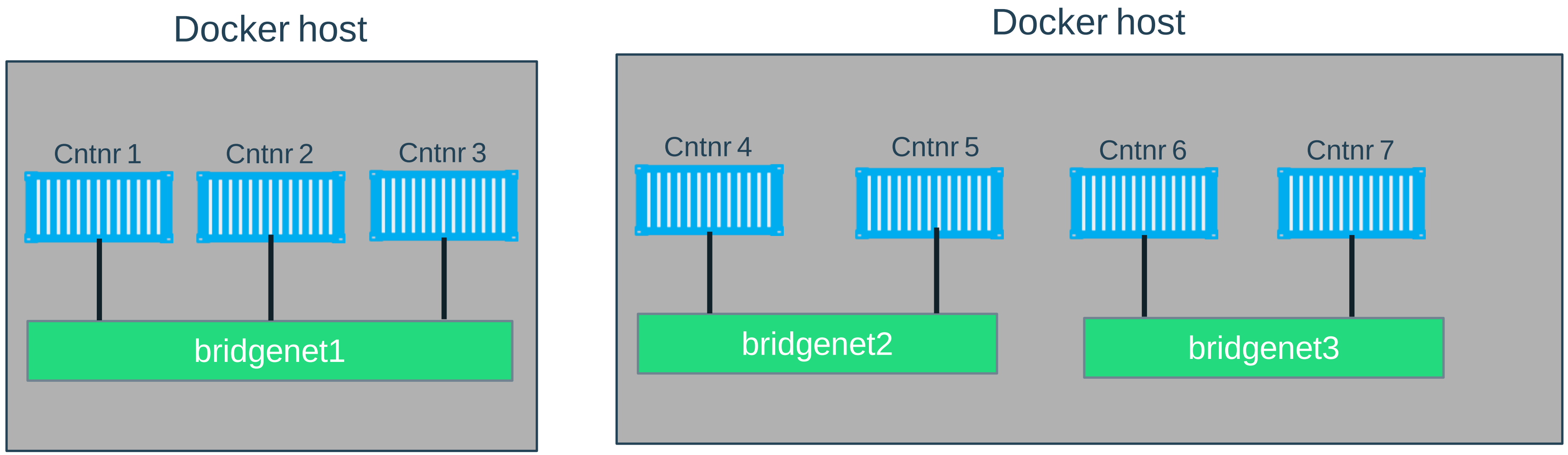
- Improve performance
  - As directory structures get complicated traversing the tree can slow system performance

- Data persistence

Section 3:
Networking

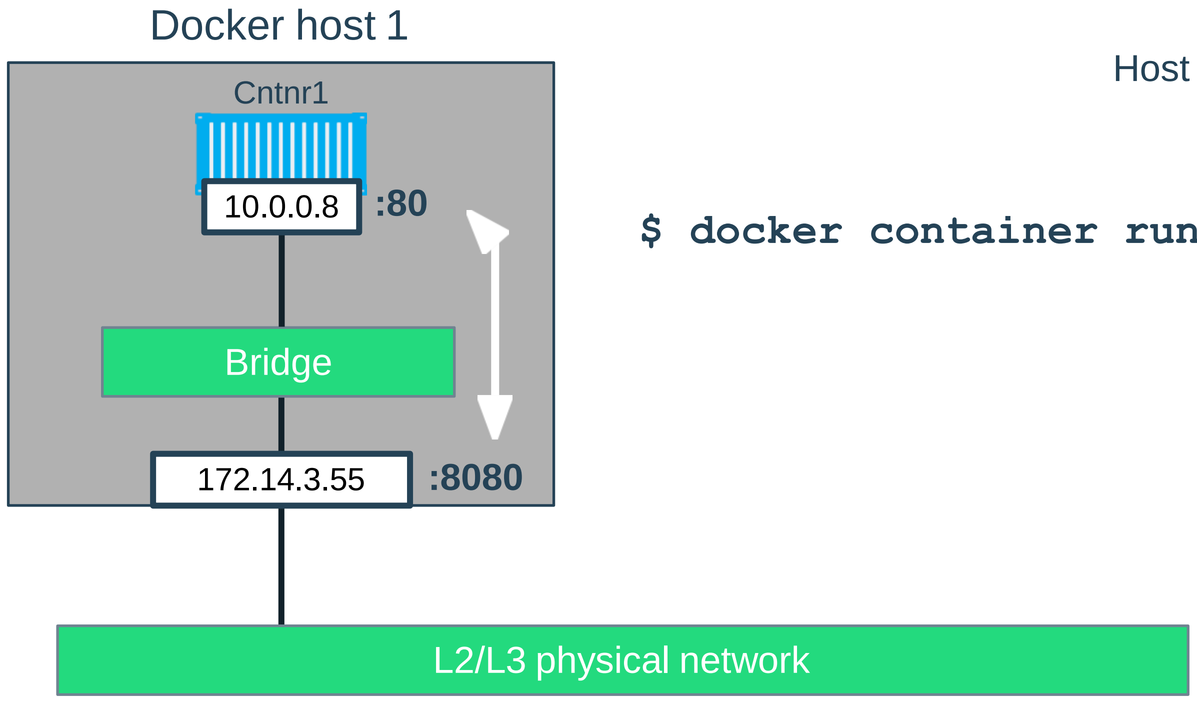
docker

What is Docker Bridge Networking



```
docker network create -d bridge --name bridgenet1
```

Docker Bridge Networking and Port Mapping



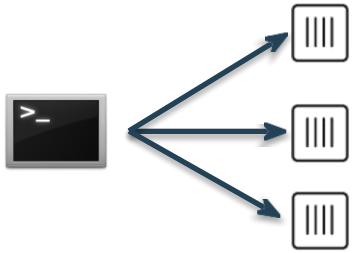
Host port
Container port

```
$ docker container run -p 8080:80 ...
```

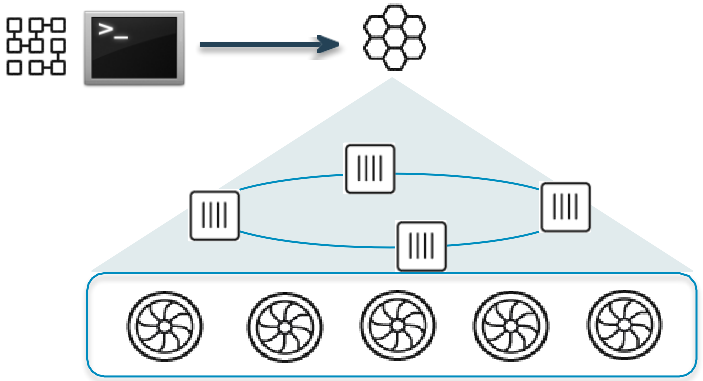
Section 4:

Docker Compose: Multi Container Applications

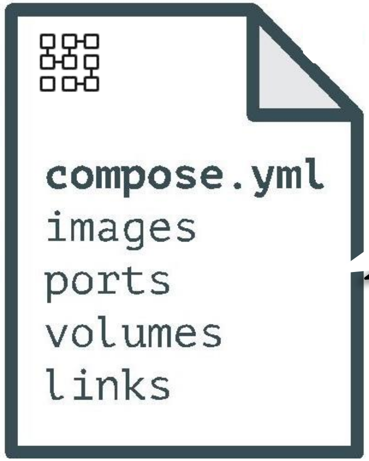
- Build and run one container at a time
- Manually connect containers together
- Must be careful with dependencies and start up order



- Define multi container app in compose.yml file
- Single command to deploy entire app
- Handles container dependencies
- Works with Docker Swarm, Networking, Volumes, Universal Control Plane



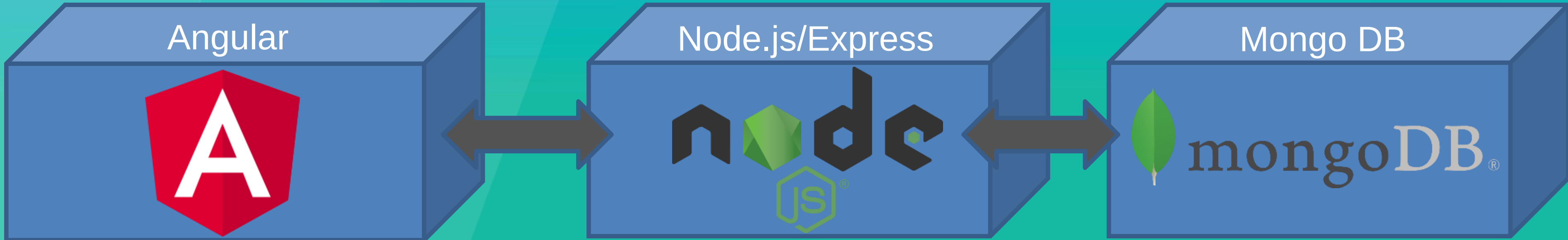
Docker Compose: Multi Container Applications



```
version: '2' # specify docker-compose version

# Define the services/containers to be run
services:
angular: # name of the first service
build: client # specify the directory of the Dockerfile
ports:
- "4200:4200" # specify port forewarding

express: #name of the second service
build: api # specify the directory of the Dockerfile
ports:
- "3977:3977" #specify ports forewarding

database: # name of the third service
image: mongo # specify image to build container from
ports:
- "27017:27017" # specify port forewarding
```

Docker Compose: Scale Container Applications

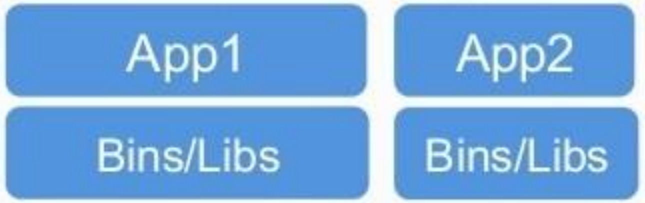
Demo



docker

Containers

What are containers?



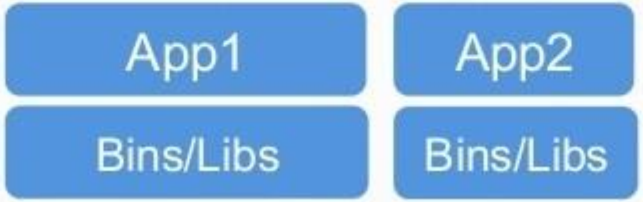
OS virtualization

Process isolation

Automation

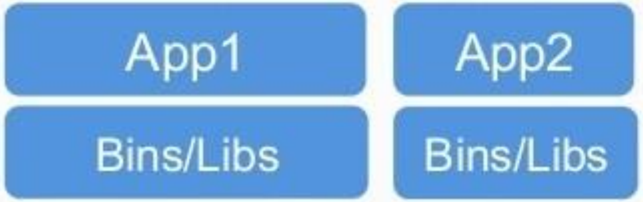
Images

Container advantages



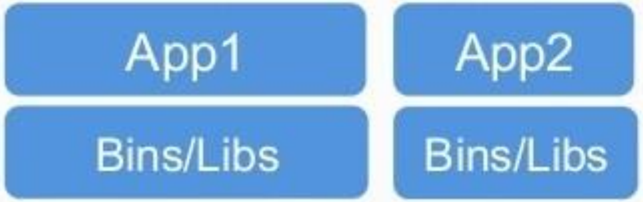
Portable

Container advantages



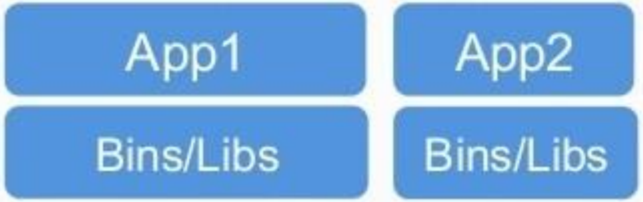
Flexible

Container advantages



Fast

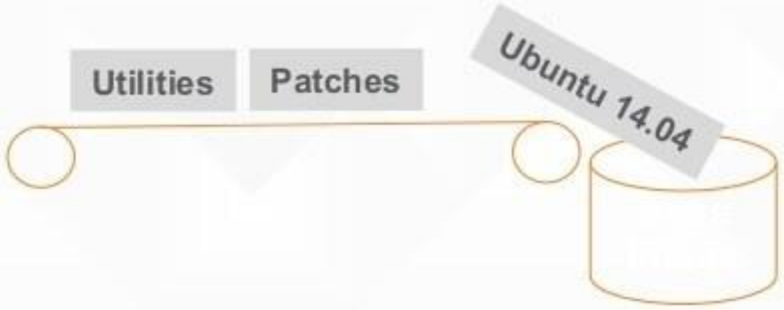
Container advantages



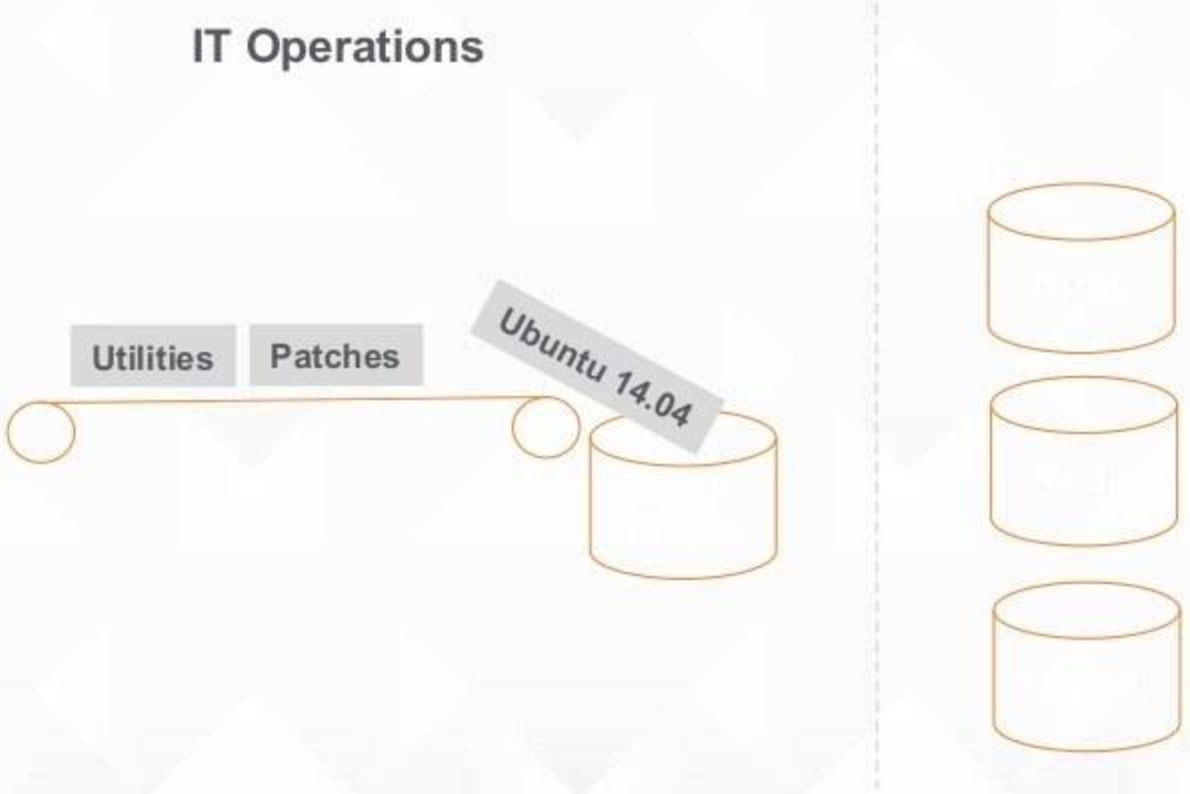
Efficient

A container pipeline

IT Operations

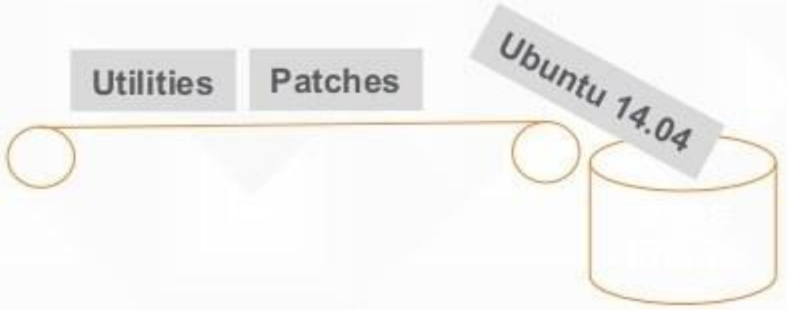


A container pipeline

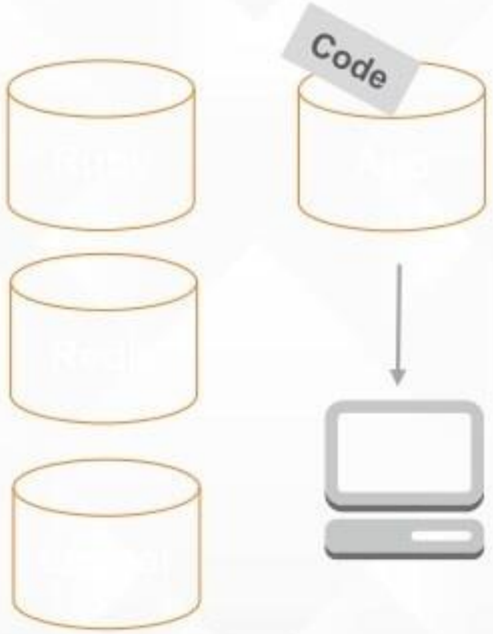


A container pipeline

IT Operations

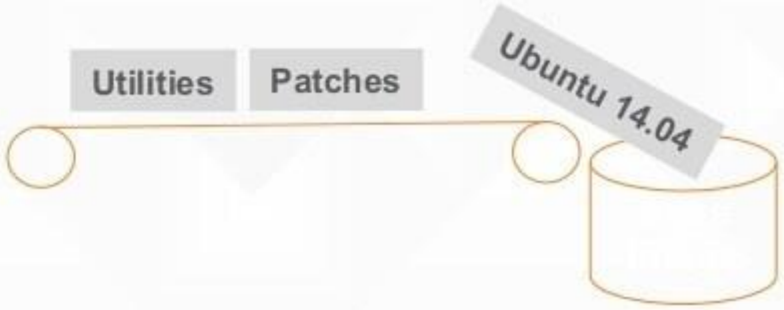


Developer

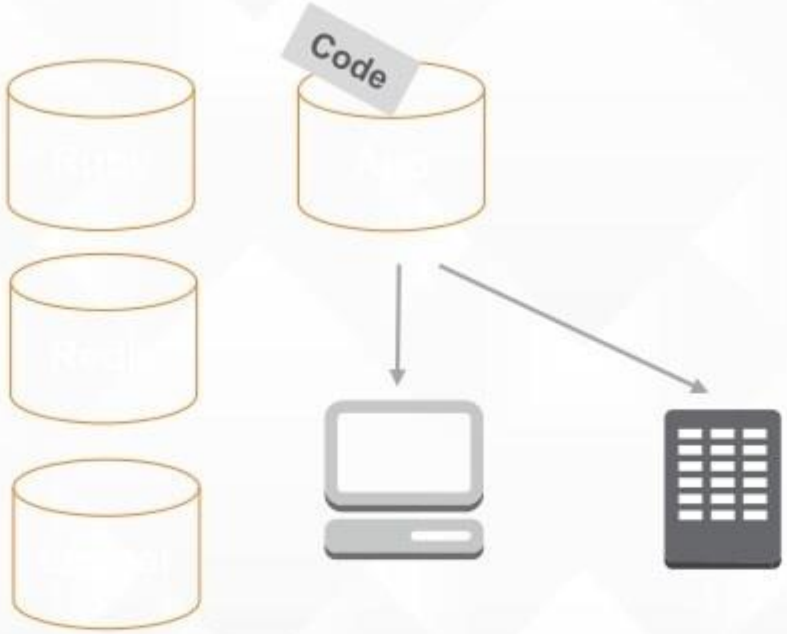


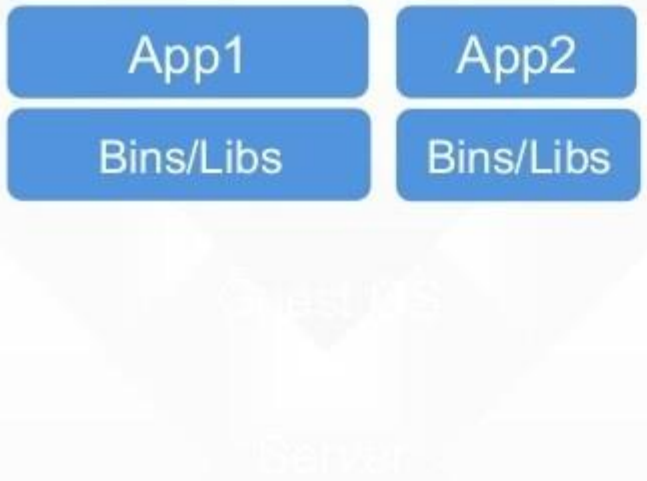
A container pipeline

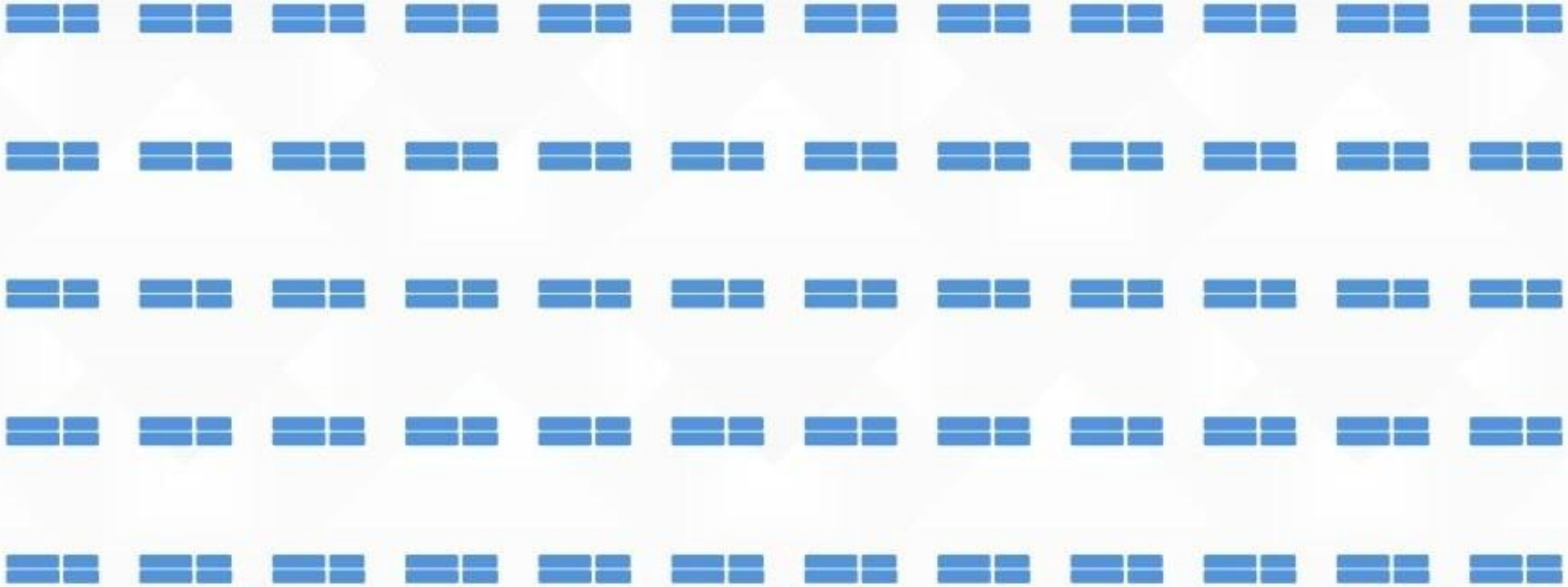
IT Operations



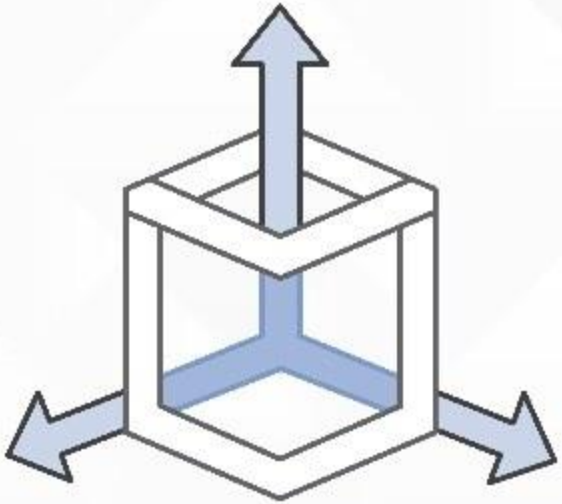
Developer




App1
App2
Bins/Libs
Bins/Libs
Guest OS
Server



Easily manage clusters for any scale



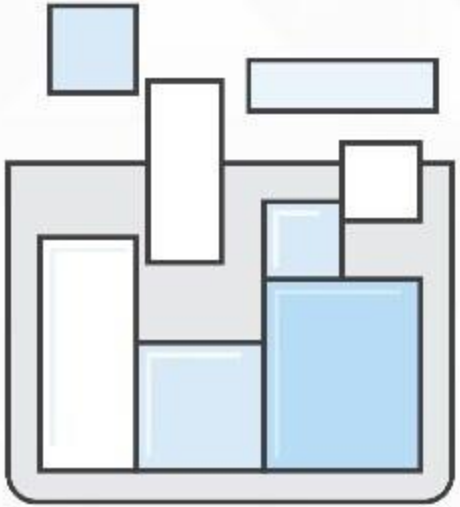
Nothing to run

Complete state

Control and monitoring

Scale

Flexible container placement

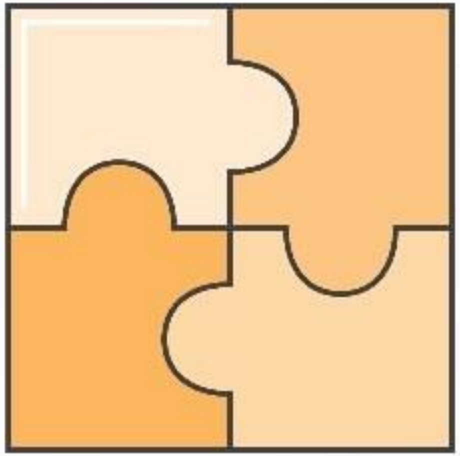


Applications

Batch jobs

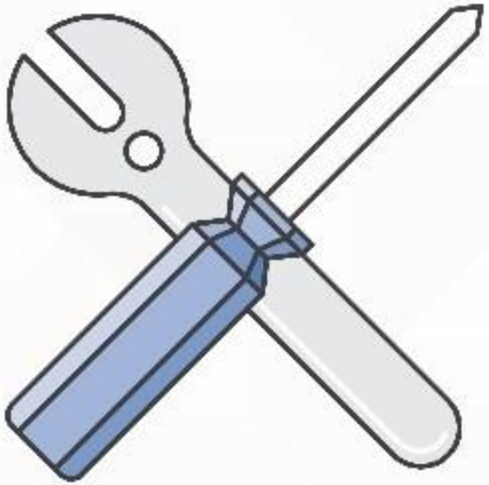
Multiple schedulers

Designed for use with other AWS services



- Elastic Load Balancing
- Amazon Elastic Block Store
- Amazon Virtual Private Cloud
- AWS Identity and Access Manageme
- AWS CloudTrail

Extensible



Comprehensive APIs

Open source agent

Custom schedulers

Common Patterns

Pattern 1: services and applications

Simple to model

Decompose to smaller (micro) services

Blue / green deployments

Pattern 2: batch jobs

Share pools of resources

APIs provide cluster state

Auto Scaling, Spot, Reserved Instances

Intro to Amazon ECS

Abby Fuller, AWS
@abbyfuller

Amazon EC2 Container Service

Amazon EC2 Container Service (ECS)

Highly scalable, high performance container management system.

Eliminates the need to install, operate, and scale your own container management infrastructure.

Amazon EC2 Container Service (ECS)

ECS provides a managed platform for:

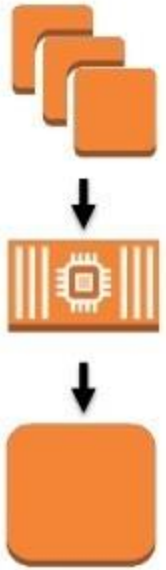
Cluster management

Container orchestration

Deep AWS integration

@abbyfulle

How does ECS map to traditional workloads?

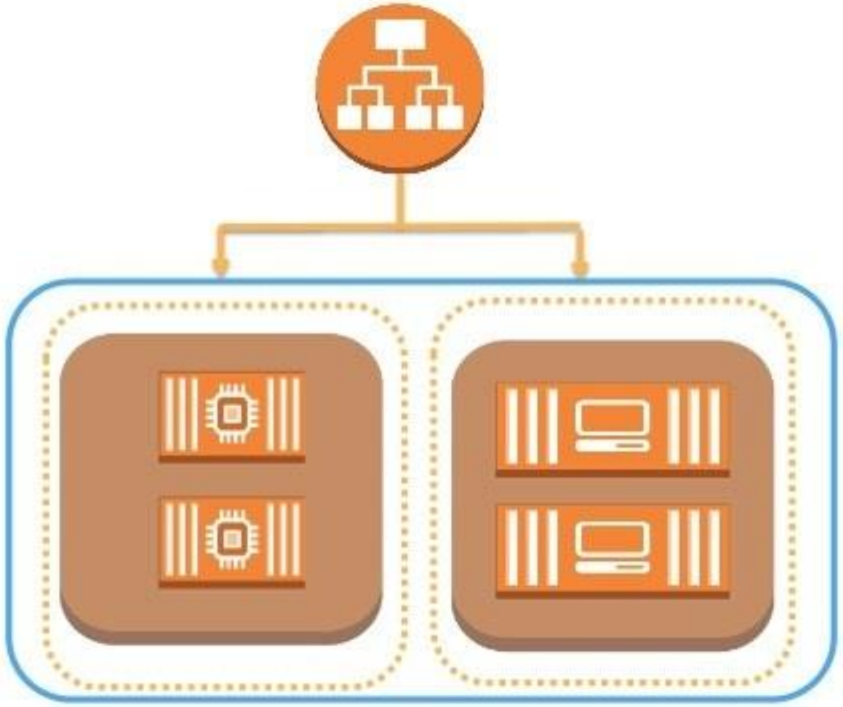


Instances: standard EC2 boxes. Once registered to a Cluster, your Tasks run here

Services: layer that manages and places Tasks

Tasks: container wrapper and configuration around processes running on the instance

How does ECS work?



Load balancer: (ALB or EC2 classic) routes traffic to the cluster instances.

Cluster is made up of one or more EC2 instances

Each **cluster instance** runs one or more **Services**

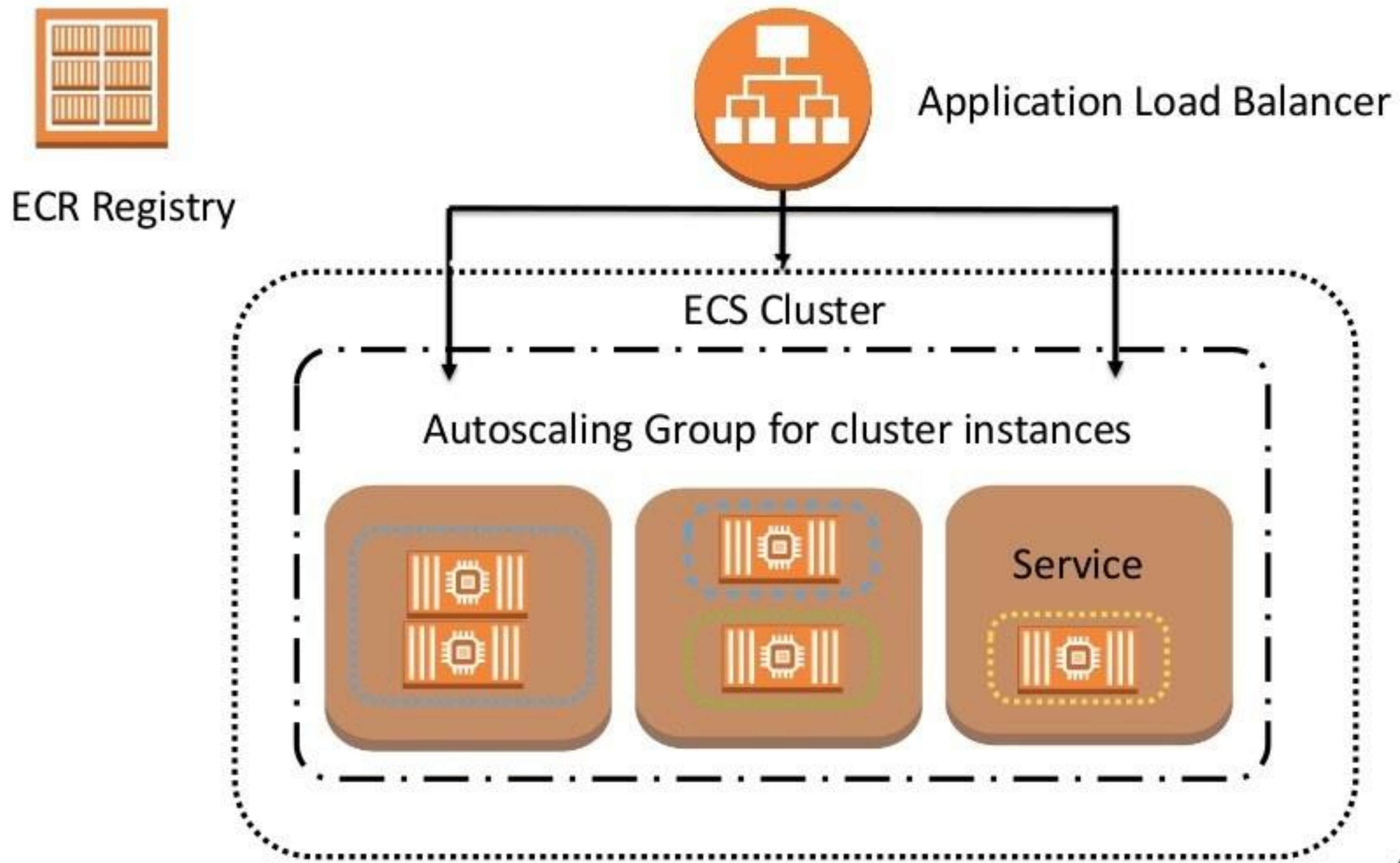
@abbyfulle

How does ECS work?

Each **cluster instance** runs one or more **Services**

A **Service** controls things like the number of copies of a Task you want running (Desired Count), and registers your Service with a load balancer

A **Task Definition** controls things like container image, environment variables, resource allocation, logger, and other parameters


ECR Registry
Application Load Balancer
ECS Cluster
Autoscaling Group for cluster instances
Service

@abbyfulle

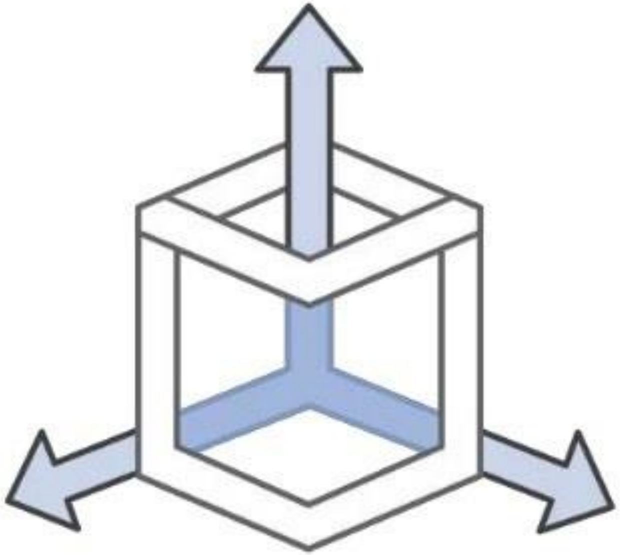
Let’s talk about ALB

- **Define routing rules based on content**. Fancy way of saying “send traffic to different services based on endpoint”. This is magical.

- As a bonus, this allows ECS to allocate ports dynamically rather than statically, and one ALB can handle multiple services.

Why ECS?

Bottom line: containers and microservices can require a lot of orchestration and moving pieces. ECS removes a lot of this heavy lifting.



@abbyfulle

Who is using ECS?

Prezi

Instacart

CapitalOne

Mapbox

shippable

air

mytaxi

Segment.i

yle

edmunds.com

coursera

Expedia

Upserve

…and many more

Let’s get (feature) specific

A few features, but many more.

Amazon ECS Task Placement

IAM Roles for Tasks

Flexible scaling for performance

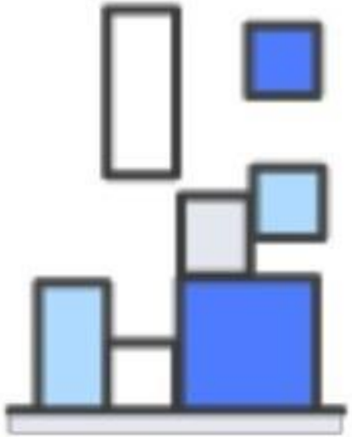
Amazon ECS Event Stream for Cloudwatch Logs

Fast, hassle-free deployments

@abbyfulle

Amazon ECS Task Placement

- A *task placement strategy* is an algorithm for selecting instances for task placement, or tasks for termination
- A *task placement constraint* is a rule taken into consideration during task placement
- Strategies and constraints can be used together



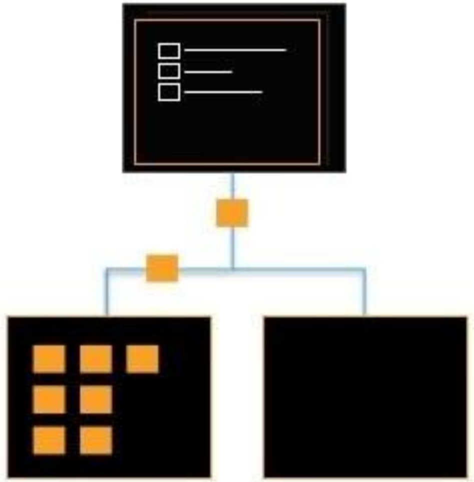
@abbyfulle

How can strategies and policies be used?

Name	Example
AMI ID	attribute:ecs.ami-id == ami-eca289fb
Availability Zone	attribute:ecs.availability-zone == us-east-1a
Instance Type	attribute:ecs.instance-type == t2.small
Distinct Instances	type="distinctInstances"
Custom	attribute:stack == prod

@abbyfulle

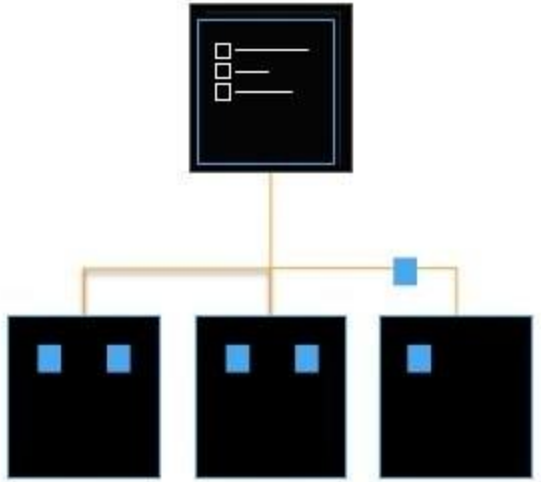
Multiple strategies are supported



Binpacking

???

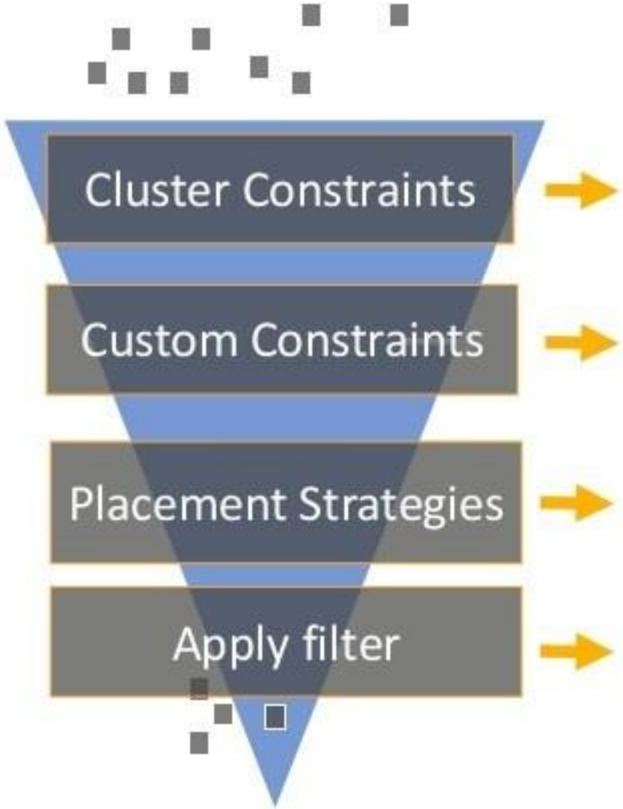
Random



Spread

@abbyfulle

How it works



Satisfy CPU, memory, and port requirements

Filter for location, instance-type, AMI, or custo
attribute constraints

Identify instances that meet spread or binpack
placement strategy

Select final container instances for placement

@abbyfulle

Amazon ECS Event Stream for Cloudwatch Logs

- Receive near real-time updates about both the current state of both the container instances within the ECS Cluster, and the current state of all tasks running on those container instances.
- Can be used to build custom schedulers, or to monitor cluster state and handle those state changes by consuming events with other AWS services, such as Lambda.

IAM Roles for ECS Tasks

- **Specify an IAM role used by the containers in a task.**
- Credential Isolation: containers can only access the role for the specific task that they are assigned to.
- **Authorization:** Unauthorized containers cannot access IAM role credentials defined for other tasks.
- **Auditability:** Audit through CloudTrail. Can track the Task credentials taskARN to show which task is using which role.

Fast, hassle-free deployments

- **Services deploy and scale quickly**. Very easily extensible through API calls; for example, trigger a deployment based on a commit to a branch on Github through your CI tool.
- **Plus, extra protection baked in**. ECS will only drain connections from the previous Task Definition if the new Task Definition passes health checks.

Flexible scaling for performance

- **Scale a service up or down based on CloudWatch alarms.** Autoscaling is built into the Service during the registration process.
- Since Clusters are part of EC2 Autoscaling Groups, you can also **scale the Cluster itself based on resources**, like you would any other group.

A great disturbance in the force

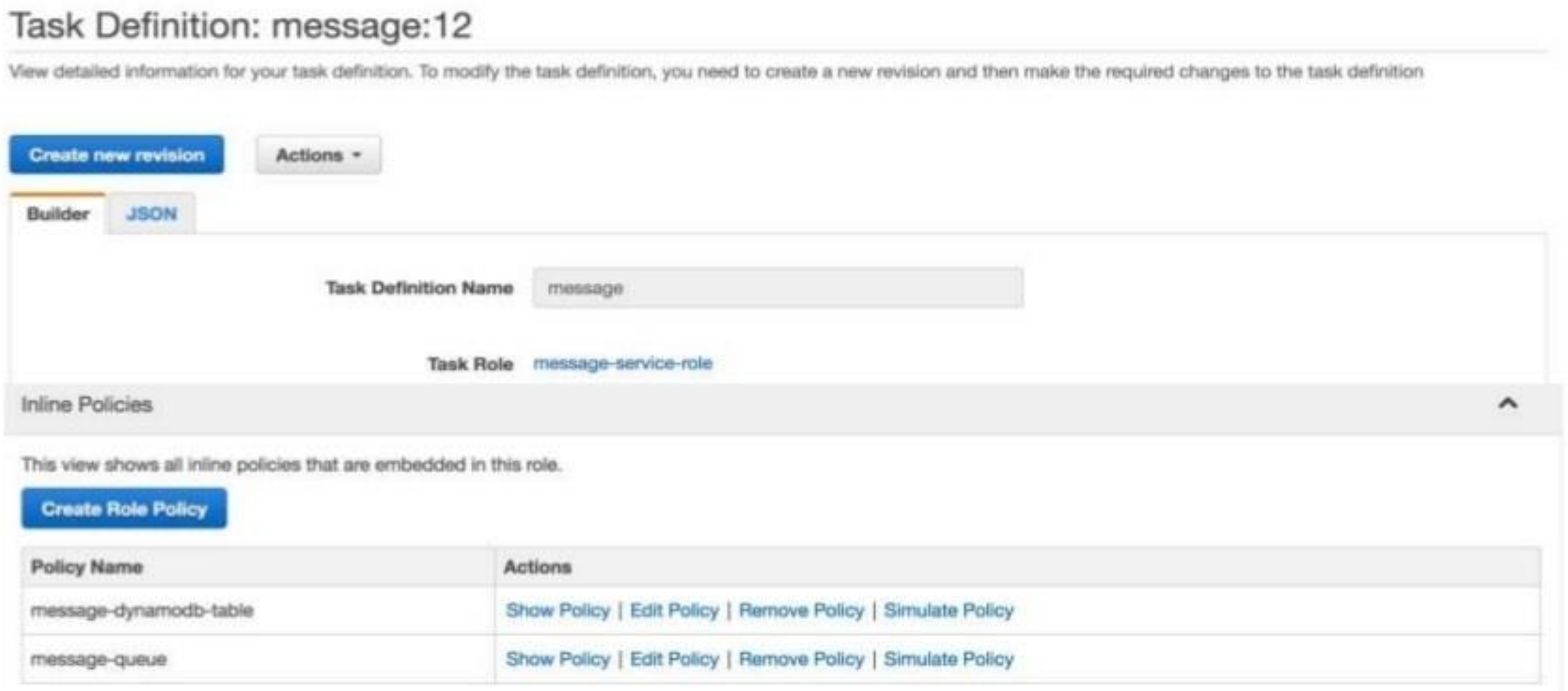
- With the shift to microservices, comes a shift in thinking: more and more options are moving from just the server level to the containers themselves.
- Don't just move a service over to containers and call it a day: decompose and rebuild.
- Security (IAM), scaling (Task-level autoscaling), traffic distribution (ALB and NLB), configuration, settings → all happening at the container/service level now.

With more services comes more responsibility

- More moving pieces
- Safety and security first
- Choose the right option (tool, language, setting) that works for you.
- Use your resources! Document, alert, automate.

@abbyfulle

IAM Roles for Tasks



Task Definition: message:12

View detailed information for your task definition. To modify the task definition, you need to create a new revision and then make the required changes to the task definition

Task Definition Name: message

Task Role: message-service-role

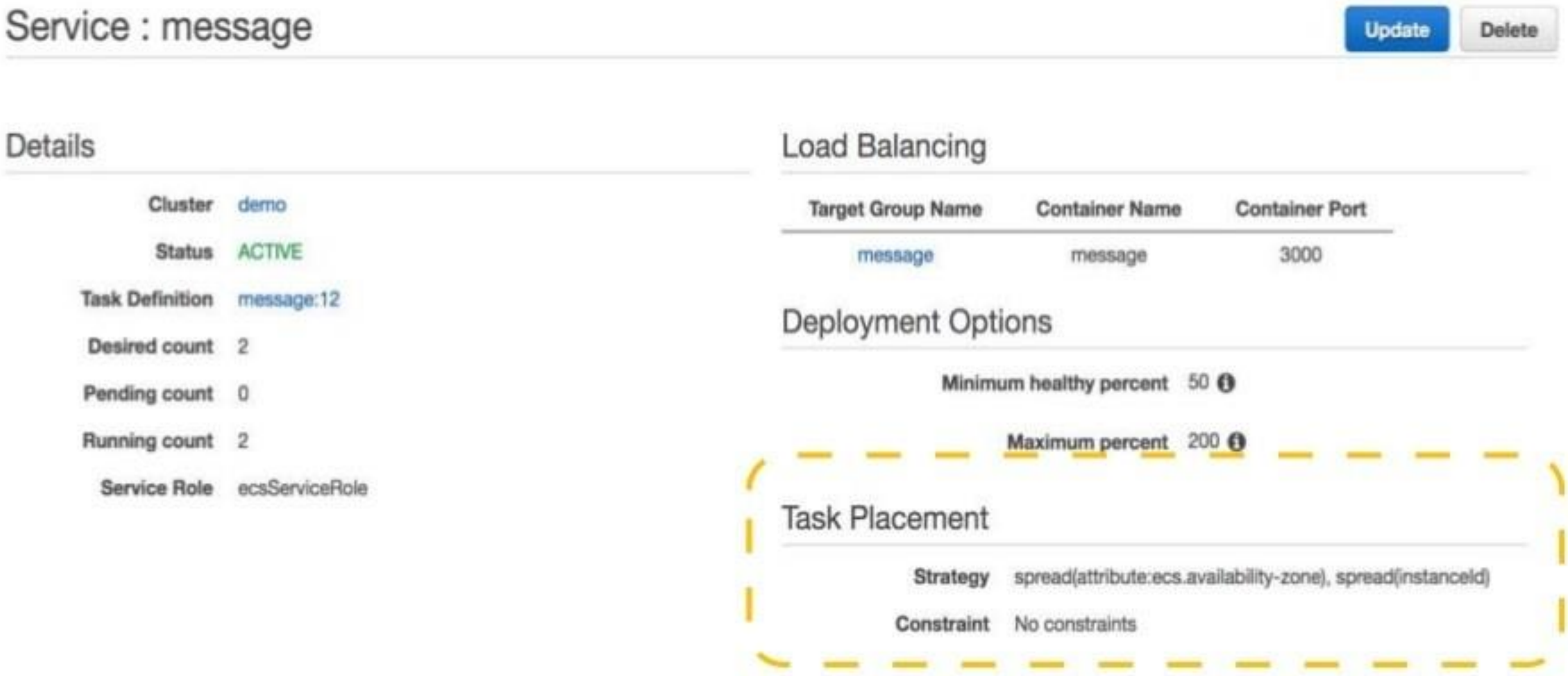
Inline Policies

This view shows all inline policies that are embedded in this role.

Policy Name	Actions
message-dynamodb-table	Show Policy | Edit Policy | Remove Policy | Simulate Policy
message-queue	Show Policy | Edit Policy | Remove Policy | Simulate Policy

@abbyfulle

Amazon ECS Task Placement



Service : message

Details

Cluster	demo
Status	ACTIVE
Task Definition	message:12
Desired count	2
Pending count	0
Running count	2
Service Role	ecsServiceRole

Load Balancing

Target Group Name	Container Name	Container Port
message	message	3000

Deployment Options

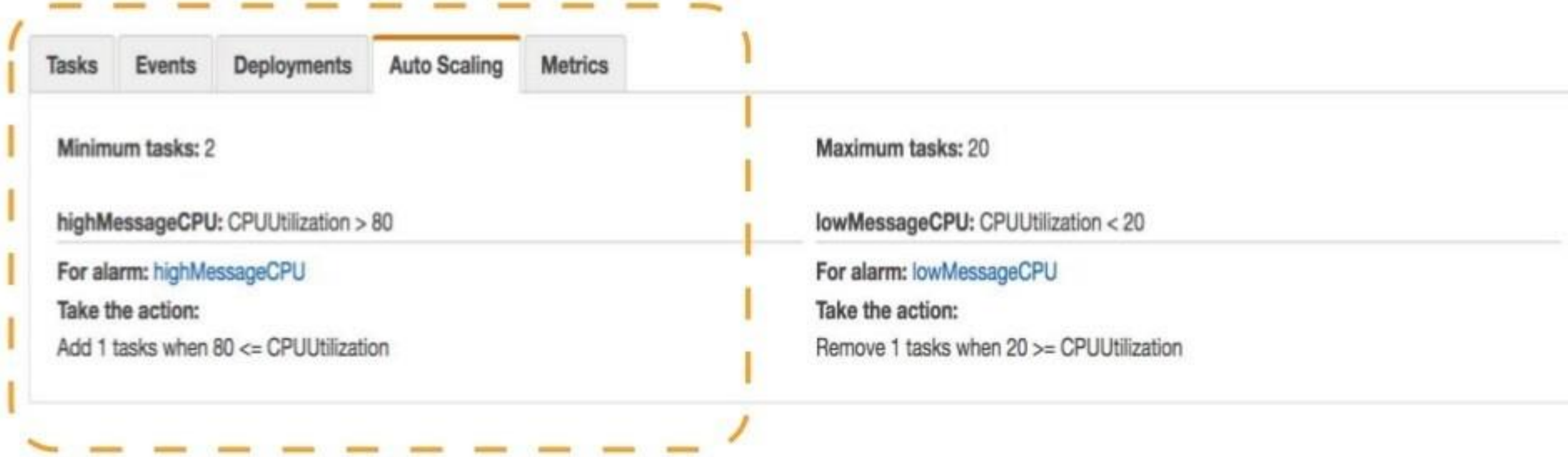
Minimum healthy percent	50
Maximum percent	200

Task Placement

Strategy	spread(attribute:ecs.availability-zone), spread(instanceId)
Constraint	No constraints

@abbyfulle

Autoscaling



Minimum tasks: 2

highMessageCPU: CPUUtilization > 80

For alarm: highMessageCPU

Take the action:

Add 1 tasks when 80 <= CPUUtilization

Maximum tasks: 20

lowMessageCPU: CPUUtilization < 20

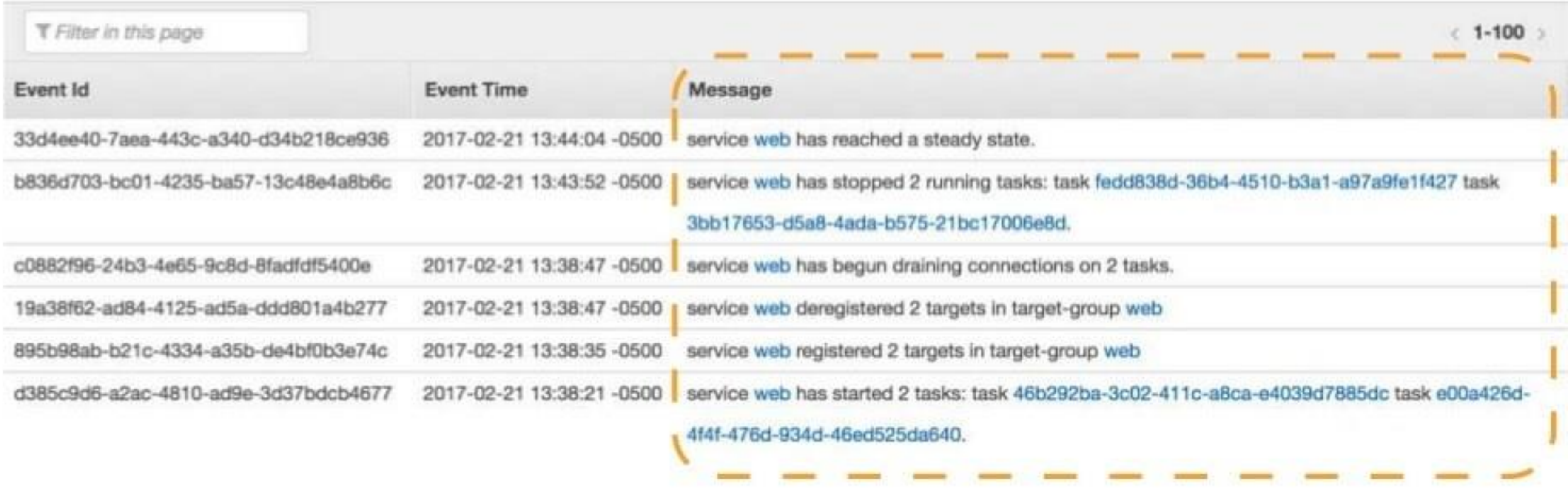
For alarm: lowMessageCPU

Take the action:

Remove 1 tasks when 20 >= CPUUtilization

@abbyfulle

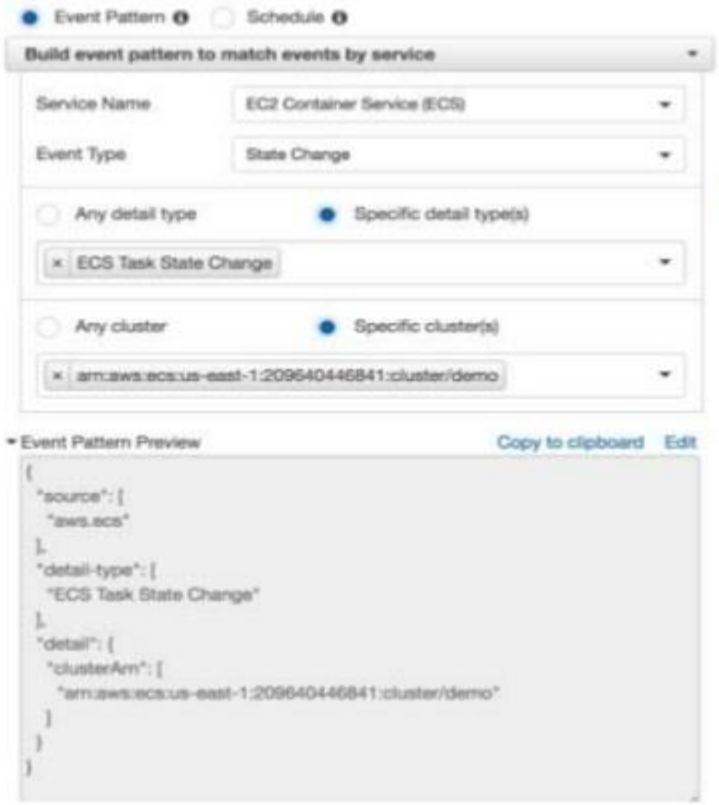
Deployments



Event Id	Event Time	Message
33d4ee40-7aea-443c-a340-d34b218ce936	2017-02-21 13:44:04 -0500	service web has reached a steady state.
b836d703-bc01-4235-ba57-13c48e4a8b6c	2017-02-21 13:43:52 -0500	service web has stopped 2 running tasks: task fedd838d-36b4-4510-b3a1-a97a9fe1f427 task 3bb17653-d5a8-4ada-b575-21bc17006e8d.
c0882f96-24b3-4e65-9c8d-8fadfdf5400e	2017-02-21 13:38:47 -0500	service web has begun draining connections on 2 tasks.
19a38f62-ad84-4125-ad5a-ddd801a4b277	2017-02-21 13:38:47 -0500	service web deregistered 2 targets in target-group web
895b98ab-b21c-4334-a35b-de4bf0b3e74c	2017-02-21 13:38:35 -0500	service web registered 2 targets in target-group web
d385c9d6-a2ac-4810-ad9e-3d37bdcb4677	2017-02-21 13:38:21 -0500	service web has started 2 tasks: task 46b292ba-3c02-411c-a8ca-e4039d7885dc task e00a426d-4f4f-476d-934d-46ed525da640.

@abbyfulle

Amazon ECS Event Stream for CloudWatch



● Event Pattern   ○ Schedule

Build event pattern to match events by service

Service Name: EC2 Container Service (ECS)

Event Type: State Change

○ Any detail type   ● Specific detail type(s)

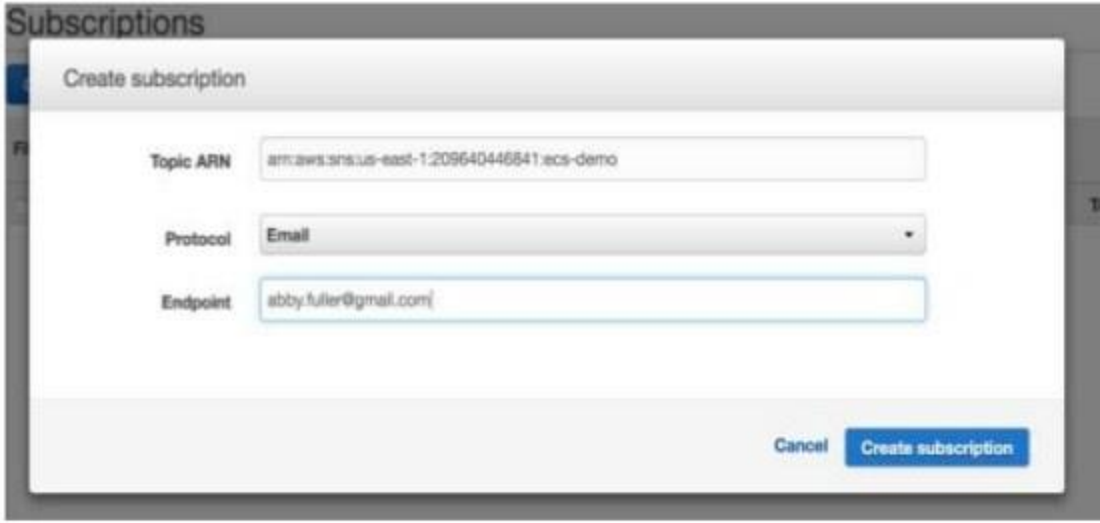
ECS Task State Change

○ Any cluster   ● Specific cluster(s)

arn:aws:ecs:us-east-1:209640446841:cluster/demo

Event Pattern Preview   Copy to clipboard   Edit

```
{
  "source": [
    "aws.ecs"
  ],
  "detail-type": [
    "ECS Task State Change"
  ],
  "detail": {
    "clusterArn": [
      "arn:aws:ecs:us-east-1:209640446841:cluster/demo"
    ]
  }
}
```



Subscriptions

Create subscription

Topic ARN: arn:aws:sns:us-east-1:209640446841:ecs-demo

Protocol: Email

Endpoint: abby.fuller@gmail.com

@abbyfulle

Some ECS resources

- AWS docs: https://aws.amazon.com/ecs/
- ECS first run wizard: https://console.aws.amazon.com/ecs/home?region=us-east-1
- Nathan Peck's ECS repo: https://github.com/nathanpeck/awesome-ecs
- More talks of mine: https://aws.amazon.com/evangelists/abby-fuller/
- ECS "Getting Started" workshop: https://www.github.com/abby-fuller/ecs-demo

Questions?