

# Introducing AWS Fargate

Running Containers without Infrastructure

Archana Srikanta, Senior Software Engineer

23<sup>rd</sup> January, 2018

# AGENDA

## Motivation

Why did we build Fargate?

## Working with Fargate

Construct a sample app on Fargate

## Demo

# MOTIVATION

At first there was



Amazon EC2

# Then Docker!

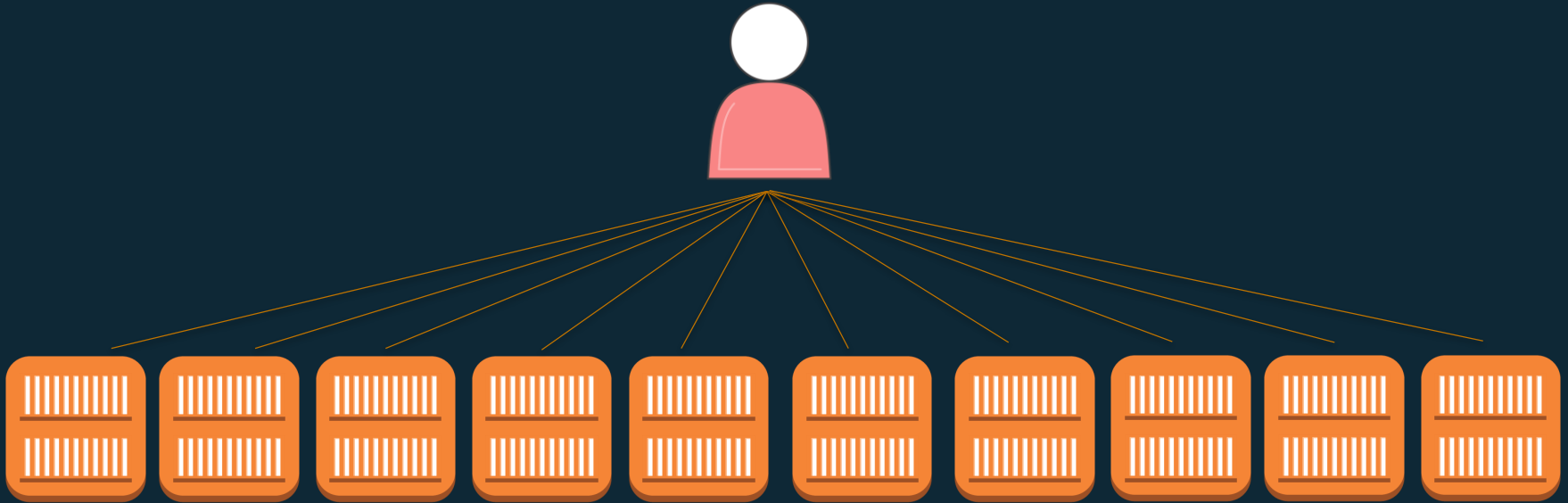


Customers started containerizing applications  
within EC2 instances

# Containers made it easy to build and scale cloud-native applications



Customers needed an easier way to manage large clusters of instances and containers



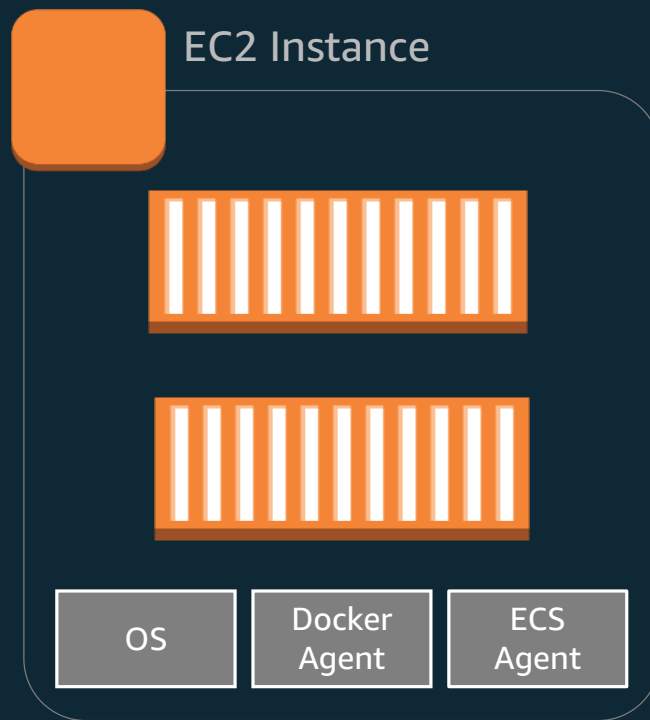
# AMAZON ELASTIC CONTAINER SERVICE

Cluster Management as a hosted service





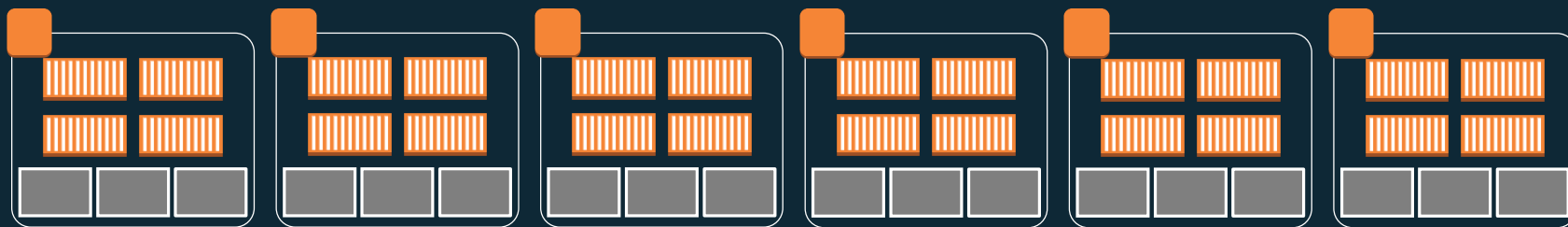
But cluster management is only half the equation...



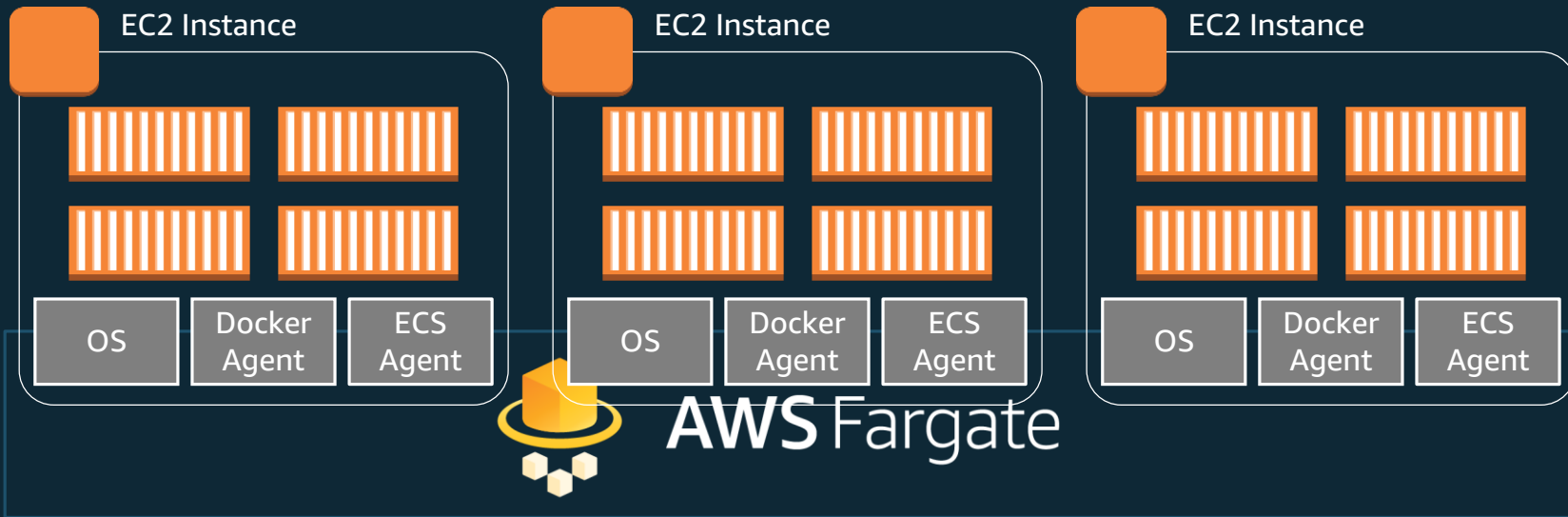
# Managing instance fleets is hard work too!

Patching and Upgrading OS, agents, etc.

Scaling the instance fleet for optimal utilization



# Customers wanted to run containers without having to manage EC2 instances



# AWS FARGATE



**Your Docker  
Containers**

## **MANAGED BY AWS**

No EC2 Instances to provision, scale or manage

## **ELASTIC**

Scale up & down seamlessly. Pay only for what you use

## **INTEGRATED**

with the AWS ecosystem: VPC Networking,  
Elastic Load Balancing, IAM Permissions, Cloudwatch and more.

# AWS Container Services Landscape

## MANAGEMENT

Deployment, Scheduling,  
Scaling & Management of  
containerized applications



Amazon Elastic  
Container Service



Amazon Elastic  
Container Service  
for Kubernetes

Preview!

## HOSTING

Where the containers run



Amazon EC2



AWS Fargate

## IMAGE REGISTRY

Container Image Repository

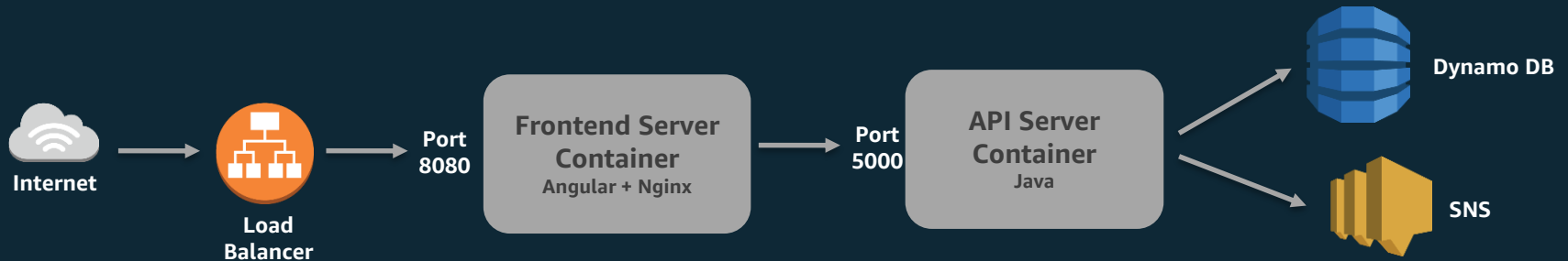


Amazon Elastic  
Container Registry

# WORKING WITH FARGATE

# WE WILL BUILD

- A TicTacToe game application, called Scorekeep on Fargate



- Configure it step by step : Compute, Networking, Storage, Permissions, Logging
- And run it!

# FARGATE CONSTRUCTS

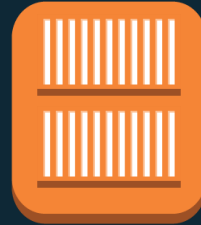
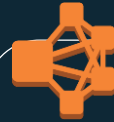


# CONSTRUCTS



register  
**Task Definition**

Define application containers: Image URL, CPU & Memory requirements, etc.



run  
**Task**

- A running instantiation of a task definition
- Use FARGATE launch type



Elastic Load Balancer



create  
**Service**

- Maintain n running copies
- Integrated with ELB
- Unhealthy tasks automatically replaced

create  
**Cluster**

- Infrastructure Isolation boundary
- IAM Permissions boundary

# TASK DEFINITION

## Task Definition Snippet

```
{
  "family": "scorekeep",
  "containerDefinitions": [
    {
      "name": "scorekeep-frontend",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/fe"
    },
    {
      "name": "scorekeep-api",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/api"
    }
  ]
}
```

Immutable, versioned document

Identified by family:version

Contains a list of up to 10 container definitions

All containers are co-located on the same host

Each container definition has:

- A name
- Image URL (ECR or Public Images)
- And more...stay tuned!

# REGISTRY SUPPORT

Amazon Elastic Container Registry (ECR)

---



Public Repositories supported

---



3<sup>rd</sup> Party Private Repositories (coming soon!)



# COMPUTE

# CPU & MEMORY SPECIFICATION

## Units

- CPU : cpu-units. 1 vCPU = 1024 cpu-units
- Memory : MB

## Task Level Resources:

- Total Cpu/Memory across all containers
- Required fields
- Billing axis

## Container Level Resources:

- Defines sharing of task resources among containers
- Optional fields

## Task Definition Snippet

```
{
  "family": "scorekeep",
  "cpu": "1 vcpu",
  "memory": "2 gb",
  "containerDefinitions": [
    {
      "name": "scorekeep-frontend",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/fe",
      "cpu": 256,
      "memoryReservation": 512
    },
    {
      "name": "scorekeep-api",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/api",
      "cpu": 768,
      "memoryReservation": 512
    }
  ]
}
```

**Task  
Level  
Resources**

**Container  
Level  
Resources**

# TASK CPU MEMORY CONFIGURATIONS

CPU	Memory
256 (.25 vCPU)	512MB, 1GB, 2GB
512 (.5 vCPU)	1GB, 2GB, 3GB, 4GB
1024 (1 vCPU)	2GB, 3GB, 4GB, 5GB, 6GB, 7GB, 8GB
2048 (2 vCPU)	Between 4GB and 16GB in 1GB increments
4096 (4 vCPU)	Between 8GB and 30GB in 1GB increments

50 different CPU/Memory configurations to choose from

# PRICING

Pay for what you provision

---

Billed for Task level CPU and Memory

---

Per-second billing. 1 minute minimum

---

# NETWORKING



# VPC INTEGRATION

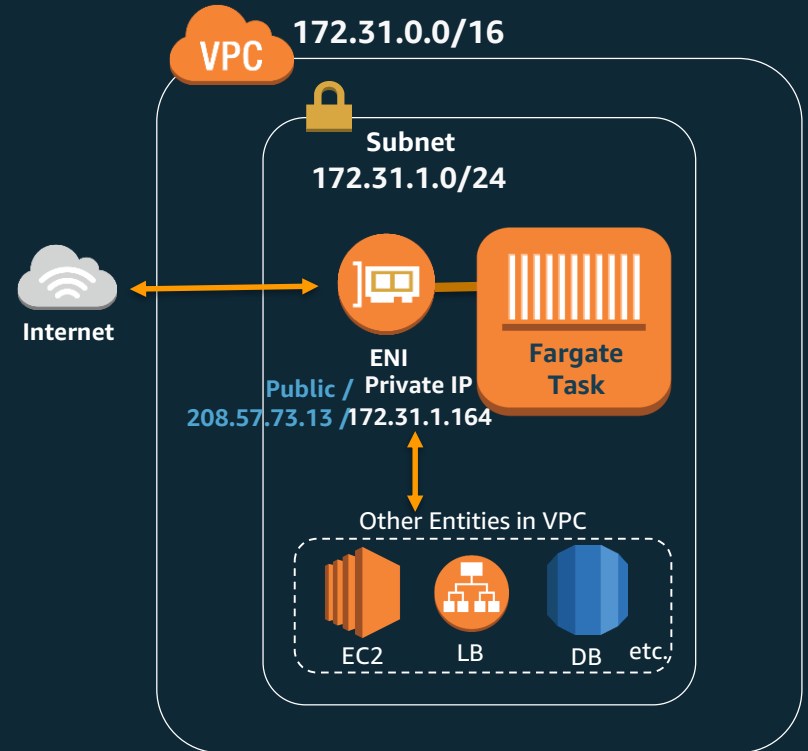
Launch your Fargate Tasks into subnets

Under the hood :

- We create an Elastic Network Interface (ENI)
- The ENI is allocated a private IP from your subnet
- The ENI is attached to your task
- Your task now has a private IP from your subnet!

You can assign public IPs to your tasks

Configure security groups to control inbound & outbound traffic



# VPC CONFIGURATION

## Task Definition

```
{
  "family": "scorekeep",
  "cpu": "1 vCpu",
  "memory": "2 gb",
  "networkMode": "awsvpc",
  "containerDefinitions": [
    {
      "name": "scorekeep-frontend",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/fe",
      "cpu": 256,
      "memoryReservation": 512
    },
    {
      "name": "scorekeep-api",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/api",
      "cpu": 768,
      "memoryReservation": 512
    }
  ]
}
```

Enables ENI  
creation &  
attachment  
to Task

## Run Task

```
$ aws ecs run-task ...
-- task-definition scorekeep:1
-- network-configuration
  "awsvpcConfiguration = {
    subnets=[subnet1-id, subnet2-id],
    securityGroups=[sg-id]
  }"
```

# INTERNET ACCESS

The Task ENI is used for all inbound & outbound network traffic to and from your task

It is also used for:

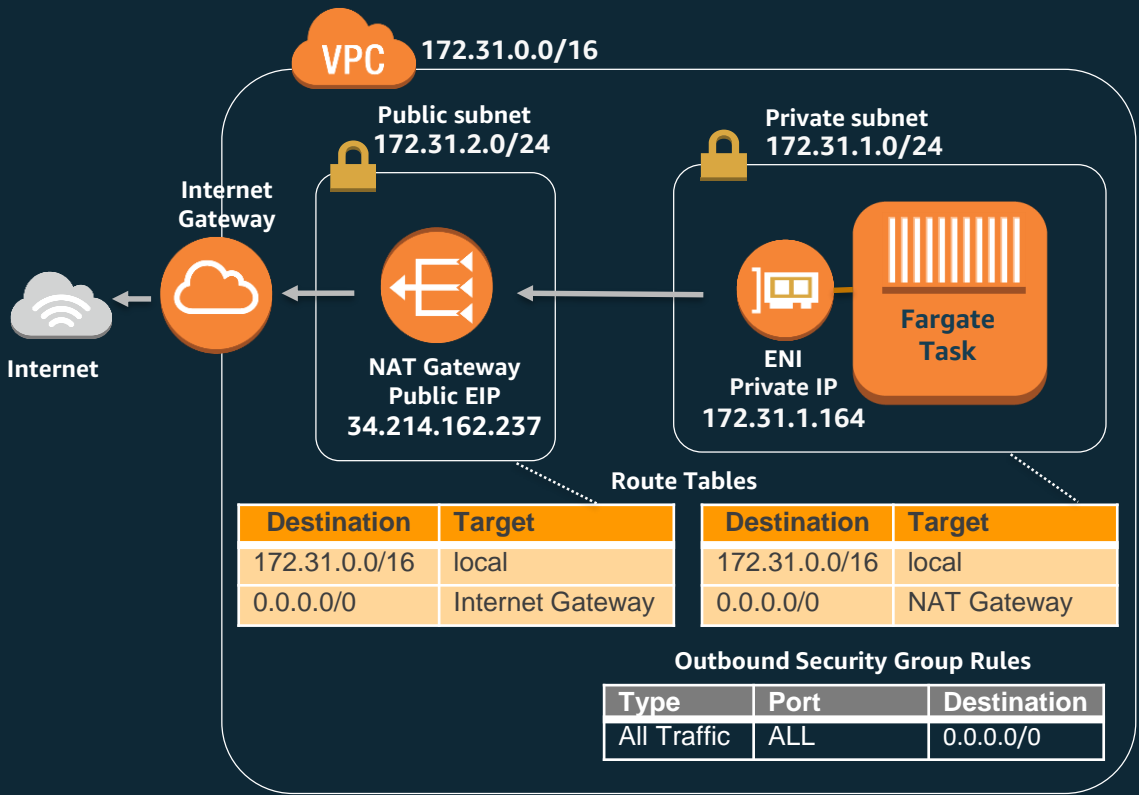
- Image Pull (from ECR or a public repository)
- Pushing logs to Cloudwatch

These endpoints need to be reachable via your task ENI

Two common modes of setup:

- Private with no inbound internet traffic, but allows outbound internet access
- Public task with both inbound and outbound internet access

# PRIVATE TASK SETUP



Attach Internet Gateway to VPC

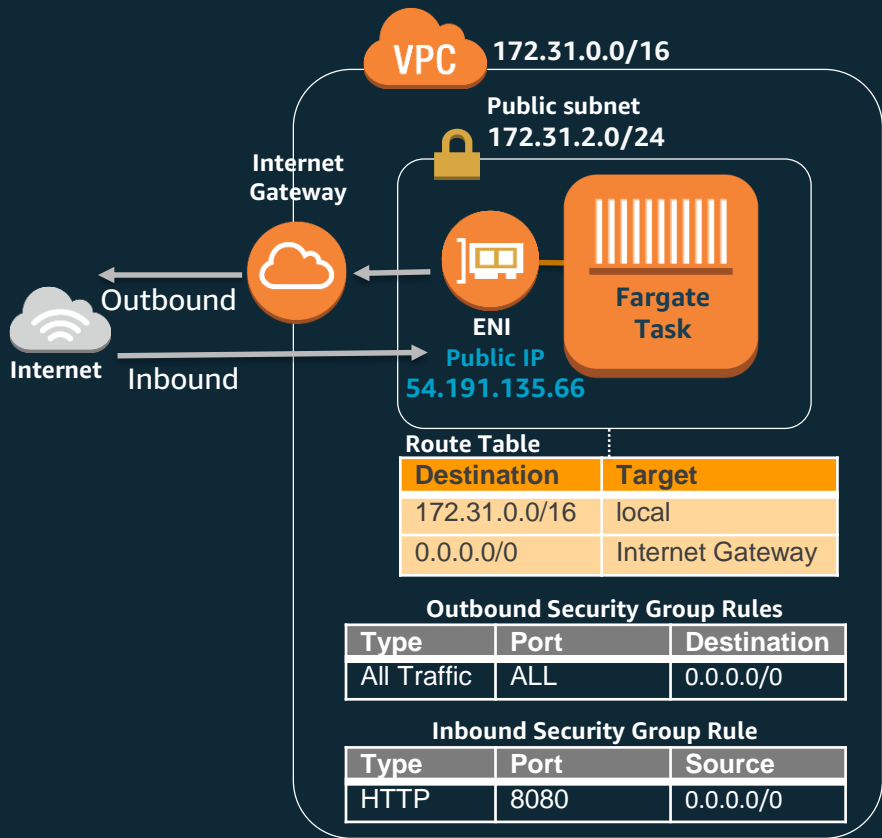
- Setup a Public Subnet with
- Route to Internet Gateway
  - NAT Gateway

- Setup Private Subnet with
- Fargate Task
  - Route to NAT Gateway

Security Group to allow outbound traffic



# PUBLIC TASK SETUP



## Run Task

```
$ aws ecs run-task ...  
  -- network-configuration  
    "awsvpcConfiguration = {  
      subnets=[public-subnet],  
      securityGroups=[sg-id],  
      assignPublicIp=ENABLED  
    }"
```

Launch the task into a Public subnet

Give it a public IP address

Security Group to allow the expected inbound traffic

# ELB CONFIGURATION

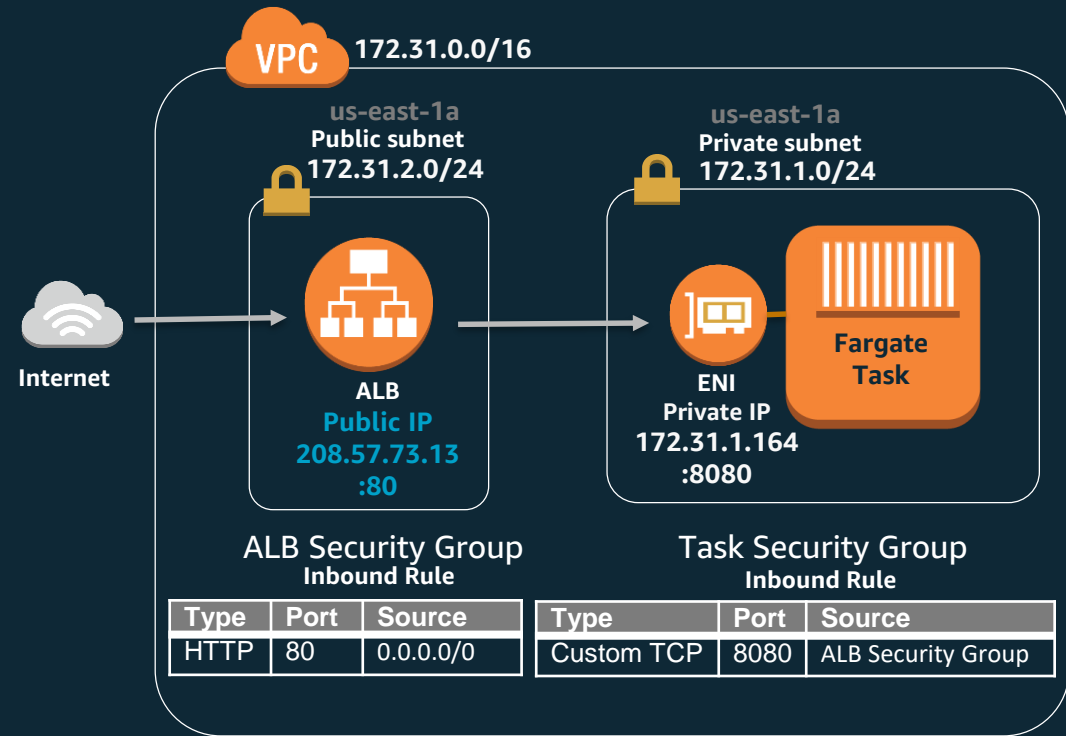
## Task Definition

```
{
  "family": "scorekeep",
  "cpu": "1 vCpu",
  "memory": "2 gb",
  "networkMode": "awsvpc",
  "containerDefinitions": [
    {
      "name": "scorekeep-frontend",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/fe",
      "cpu": 256,
      "memoryReservation": 512,
      "portMappings": [
        { "containerPort": 8080 }
      ]
    },
    {
      "name": "scorekeep-api",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/api",
      "cpu": 768,
      "memoryReservation": 512,
      "portMappings": [
        { "containerPort": 5000 }
      ]
    }
  ]
}
```

## Create Service

```
$ aws ecs create-service ...
-- task-definition scorekeep:1
-- network-configuration
    "awsvpcConfiguration = {
      subnets=[subnet-id],
      securityGroups=[sg-id]
    }"
-- load-balancers
"[
  {
    "targetGroupArn": "<insert arn>",
    "containerName": "scorekeep-frontend",
    "containerPort": 8080
  }
]"
```

# INTERNET FACING ELB VPC SETUP



Task in private subnet with private IP

ALB in public subnet with public IP

Make sure the AZs of the two subnets match

ALB security group to allow inbound traffic from internet

Task security group to allow inbound traffic from the ALB's security group

# STORAGE



# DISK STORAGE

EBS backed Ephemeral storage provided in the form of:

---

Writable Layer Storage

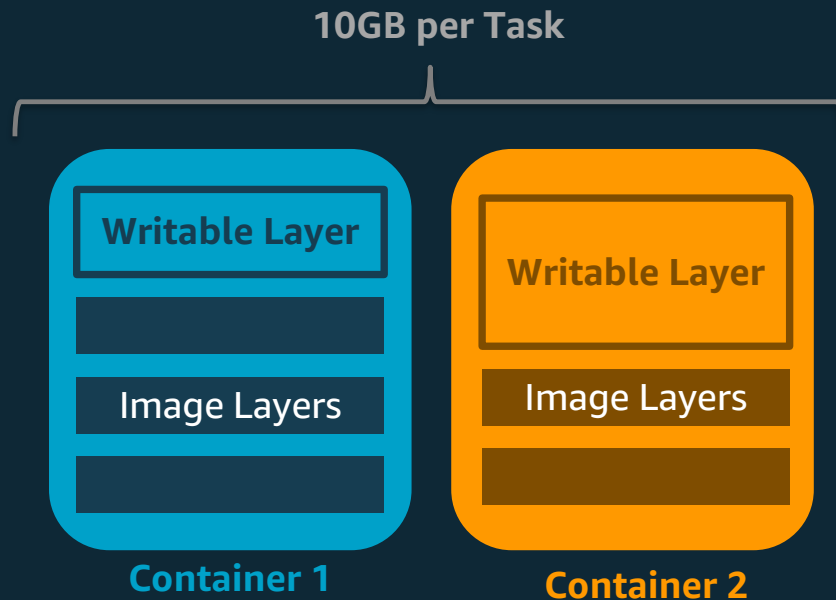
---

Volume Storage

---

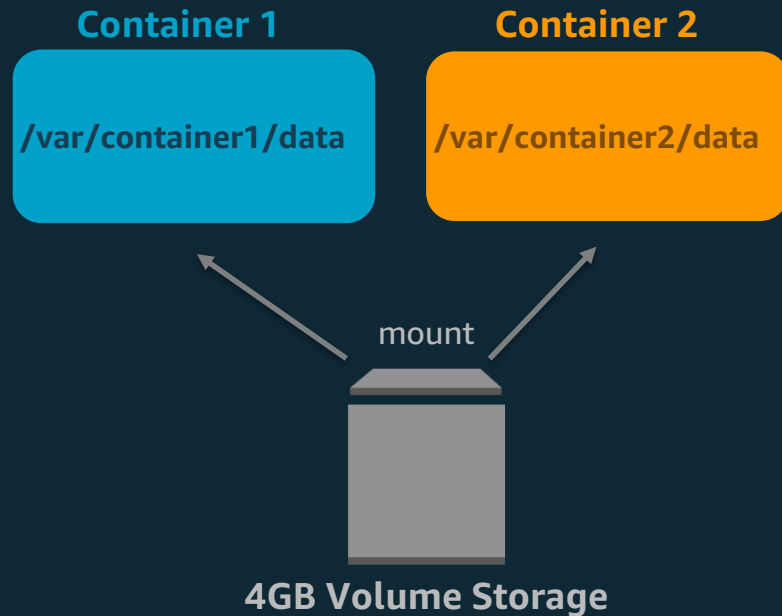
# LAYER STORAGE

- Docker images are composed of layers  
The topmost layer is the “writable” layer to capture file changes made by the running container
- 10GB Layer storage available per task, across all containers, including image layers
- Writes are not visible across containers
- Ephemeral. Storage is not available after the task stops.



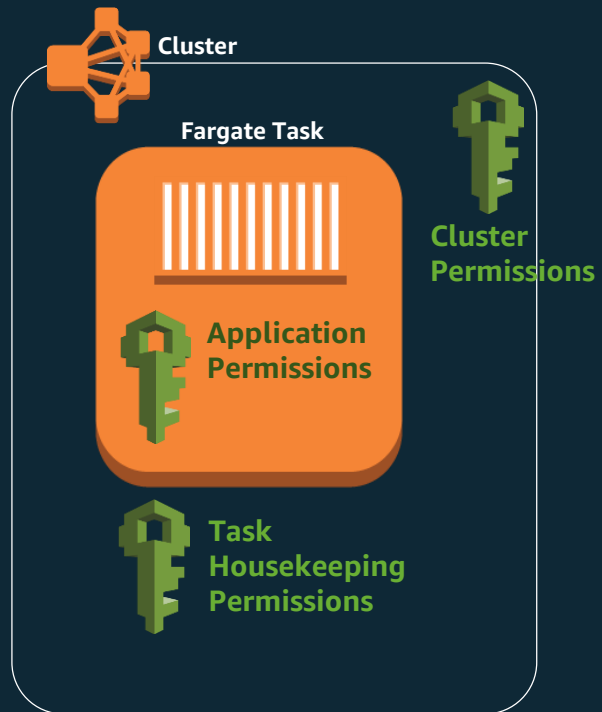
# VOLUME STORAGE

- Need writes to be visible across containers?
- Fargate provides 4GB volume space per task
- Configure via volume mounts in task definition
  - Can mount at different containerPaths
  - Do not specify host sourcePath
- Remember this is also ephemeral, i.e. not available after the task stops



# IAM PERMISSIONS

# PERMISSION TIERS



## Cluster Permissions:

Control who can launch/describe tasks in your cluster

## Application Permissions:

Allows your application containers to access AWS resources securely

## Housekeeping Permissions:

Allows us to perform housekeeping activities around your task:

- ECR Image Pull
- Cloudwatch logs pushing
- ENI creation
- Register/Deregister targets into ELB

# CLUSTER PERMISSIONS

You can tailor IAM policies for fine grained access control to your clusters  
Attach these policies to IAM Users and/or Roles as necessary

Some example policies:

```
{
  "Effect": "Allow",
  "Action": [ "ecs:RunTask" ],
  "Condition": {
    "ArnEquals": {"ecs:cluster": "<cluster-arn>"}
  },
  "Resource": [ "<task_def_family>:*" ]
}
```

Example 1: Allow RunTask in a specific cluster  
with a specific task definition only

```
{
  "Effect": "Allow",
  "Action": [ "ecs:ListTasks",
    "ecs:DescribeTasks" ],
  "Condition": {
    "ArnEquals": {"ecs:cluster": "<cluster-arn>"}
  },
  "Resource": "*"
}
```

Example 2: Read-only access to tasks in a  
specific cluster

And many more!

# APPLICATION PERMISSIONS

Do your application containers access other AWS resources?

Need to get credentials down to the task? Use a [Task Role](#)

Create an IAM Role with the requisite permissions that your application needs. In our scorekeep example, DDB & SNS permissions.

Establish a trust relationship with [ecs-tasks.amazonaws.com](https://ecs-tasks.amazonaws.com) on that role. This lets us assume the role and wire the credentials down to your task.

Add the role arn to your task definition and you're done!

We issue and rotate temporary credentials

AWS CLI/SDK calls from within your application will automatically use the Task Role credentials

## Task Definition

```
{
  "family": "scorekeep",
  "cpu": "1 vCpu",
  "memory": "2 gb",
  "networkMode": "awsvpc",
  "taskRoleArn": "arn:aws...role/scorekeepRole",
  "containerDefinitions": [
    {
      "name": "scorekeep-frontend",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/fe",
      "cpu": 256,
      "memoryReservation": 512,
      "portMappings": [
        { "containerPort": 8080 }
      ]
    },
    {
      "name": "scorekeep-api",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/api",
      "cpu": 768,
      "memoryReservation": 512,
      "portMappings": [
        { "containerPort": 5000 }
      ]
    }
  ]
}
```

# HOUSEKEEPING PERMISSIONS

- We need certain permissions in your account to bootstrap your task and keep it running.
- **Execution Role** gives us permissions for:
  - ECR Image Pull
  - Pushing Cloudwatch logs
- **ECS Service Linked Role** gives us permissions for:
  - ENI Management
  - ELB Target Registration/Deregistration



# EXECUTION ROLE

- Using an ECR Image or Cloudwatch Logs?  
Give us permissions via [an Execution Role](#)
- Create an IAM Role and add Read Permissions to ECR
  - `ecr:GetAuthorizationToken` & `ecr:BatchGetImage`
  - Or use [AmazonEC2ContainerRegistryReadOnly](#) managed policy
- Add Write Permissions to CloudWatchLogs
  - `logs:CreateLogStream` & `logs:PutLogEvents`
  - Or use [CloudWatchLogsFullAccess](#) managed policy
- Establish trust relationship with [ecs-tasks.amazonaws.com](#).  
This lets us assume the role
- Add the execution role arn into your task definition

## Task Definition

```
{
  "family": "scorekeep",
  "cpu": "1 vCpu",
  "memory": "2 gb",
  "networkMode": "awsvpc",
  "taskRoleArn": "arn:aws...role/scorekeepRole",
  "executionRoleArn":
    "arn:aws...role/scorekeepExecutionRole",
  "containerDefinitions": [
    {
      "name": "scorekeep-frontend",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/fe",
      "cpu": 256,
      "memoryReservation": 512,
      "portMappings": [
        { "containerPort": 8080 }
      ]
    },
    {
      "name": "scorekeep-api",
      "image": "xxx.dkr.ecr.us-east-1.amazonaws.com/api",
      "cpu": 768,
      "memoryReservation": 512,
      "portMappings": [
        { "containerPort": 5000 }
      ]
    }
  ]
}
```

# ECS SERVICE LINKED ROLE

- A service-linked role is a unique type of IAM role that is linked directly to an AWS service, in this case ECS
- It is automatically created in your account at first cluster creation
- It has a predefined policy, that is immutable. In this case, ENI & ELB permissions.
- You don't have to explicitly pass this role in the task definition or any API call. Just know about it in case you stumble upon it in the IAM console

# VISIBILITY & MONITORING

# CLOUDWATCH LOGS CONFIGURATION

- Use the [awslogs](#) driver to send stdout from your application to Cloudwatch logs
- Create a log group in Cloudwatch
- Configure the log driver in your task definition
- Remember to add permissions via the Task Execution Role

```
{
  "family": "scorekeep",
  ...
  "containerDefinitions": [
    {
      "name": "scorekeep-frontend",
      ...
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "scorekeep",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "scorekeep/frontend"}
        }
      },
    {
      "name": "scorekeep-api",
      ...
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "scorekeep",
          "awslogs-region": "us-east-1",
          "awslogs-stream-prefix": "scorekeep/api"}
        }
      }
    ]
  }
}
```

Task Definition

# CLOUDWATCH LOGS

View logs in the ECS or Cloudwatch Console

Clusters > default > Task: 33f9f994-9650-4ba4-ac39-644145563394

Task : 33f9f994-9650-4ba4-ac39-644145563394

Run more like this

Stop

Details

Logs

Select a container to view logs:

scopekeep-api  
scorekeep-frontend  
scopekeep-api

Last updated on November 22, 2017 5:07:58 PM (2m ago)



Logs Tab in the Task Detail Page

Timestamp (UTC+00:00)	Message
2017-11-22 11:26:41	2017-11-22 19:26:41.558 INFO 6 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'SimpleCORSFilter' to [/]
2017-11-22 11:26:41	2017-11-22 19:26:41.553 INFO 6 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean : Mapping servlet: 'dispatcherServlet' to [/]
2017-11-22 11:26:40	2017-11-22 19:26:40.258 INFO 6 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2017-11-22 11:26:40	2017-11-22 19:26:40.347 INFO 6 --- [ost-startStop-1] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 17798 ms
2017-11-22 11:26:39	2017-11-22 19:26:39.353 INFO 6 --- [ main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/8.5.4
2017-11-22 11:26:39	2017-11-22 19:26:39.351 INFO 6 --- [ main] o.apache.catalina.core.StandardService : Starting service Tomcat
2017-11-22 11:26:39	2017-11-22 19:26:39.152 INFO 6 --- [ main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialized with port(s): 5000 (http)
2017-11-22 11:26:22	2017-11-22 19:26:22.647 INFO 6 --- [ main] ationConfigEmbeddedWebApplicationContext : Refreshing org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@28...
2017-11-22 11:26:21	2017-11-22 19:26:21.357 INFO 6 --- [ main] scorekeep.Application : Starting Application with PID 6 (/scorekeep-api-1.0.0.jar started by root in /)
2017-11-22 11:26:21	2017-11-22 19:26:21.449 INFO 6 --- [ main] scorekeep.Application : No active profile set, falling back to default profiles: default
2017-11-22 11:26:19	:: Spring Boot :: (v1.4.0.RELEASE)



# OTHER VISIBILITY TOOLS

CloudWatch events on task state changes



---

Service CPU/Memory utilization metrics  
available in CloudWatch



# DEMO

# FINAL SCOREKEEP TASK DEFINITION

```
{
  "family": "scorekeep",
  "cpu": "1 vCpu",
  "memory": "2 gb",
  "networkMode": "awsipc",
  "taskRoleArn": "arn:aws:...",
  "executionRoleArn": "arn:...",
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "containerDefinitions": [...]
}
```

Task Definition

```
{
  "name": "scorekeep-frontend",
  "image": "xxx.dkr.ecr...frontend",
  "cpu": 256,
  "memoryReservation": 512,
  "portMappings": [
    { "containerPort": 8080 }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-group": "scorekeep",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix":
        "scorekeep/frontend"
    }
  }
}
```

scorekeep-frontend  
Container Definition

```
{
  "name": "scorekeep-api",
  "image": "xxx.dkr.ecr...api",
  "cpu": 768,
  "memoryReservation": 512,
  "portMappings": [
    { "containerPort": 5000 }
  ],
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-group": "scorekeep",
      "awslogs-region": "us-east-1",
      "awslogs-stream-prefix":
        "scorekeep/api"
    }
  },
  "environment": [...] #env var
}
```

scorekeep-api  
Container Definition



# LET'S RUN IT!

## Register Task Definition

```
$ aws ecs register-task-definition --cli-input-json file:///scorekeep-task-definition.json
```

## Create Cluster

```
$ aws ecs create-cluster --cluster-name FargateDemoCluster
```

## Run Task

```
$ aws ecs run-task
  --cluster FargateDemoCluster --task-definition scorekeep:1
  --launch-type FARGATE --count 1
  --network-configuration "awsvpcConfiguration={subnets=[subnet-xxx], securityGroups=[sg-xxx]}"
```

## Create Service

```
$ aws ecs create-service --service-name ScorekeepService
  --cluster FargateDemoCluster --task-definition scorekeep:1
  --launch-type FARGATE --desired-count 5
  --network-configuration "awsvpcConfiguration={subnets=[subnet-xxx], securityGroups=[sg-xxx]}"
  --load-balancer "[{"targetGroupArn":"arn:aws:xxx",
                    "containerName":"scorekeep-frontend",
                    "containerPort":8080}]"
```

Find the Fargate Scorekeep project on GitHub at

<https://github.com/awslabs/eb-java-scorekeep/tree/fargate>

# TAKE AWAYS

# TAKE AWAYS

- Fargate is a new launch type within ECS to run containers without having to manage EC2 instances
- If you're debating between EC2 v/s Fargate mode, start architecting with Fargate. It forces good design practice by keeping your application containers truly independent of the underlying host.
- If you think you must have access to the underlying host, think again.
  - There are some good reasons : special instance type needs, EC2 dedicated instances, utilizing EC2 reserved instances
  - And tell us about your use case, we want to support it on Fargate!
- Start using Fargate today!
  - Fargate works with most Docker container images
  - You can run existing task definitions on Fargate with only minor modifications.

# THANK YOU!

Learn more at [aws.amazon.com/fargate](https://aws.amazon.com/fargate)