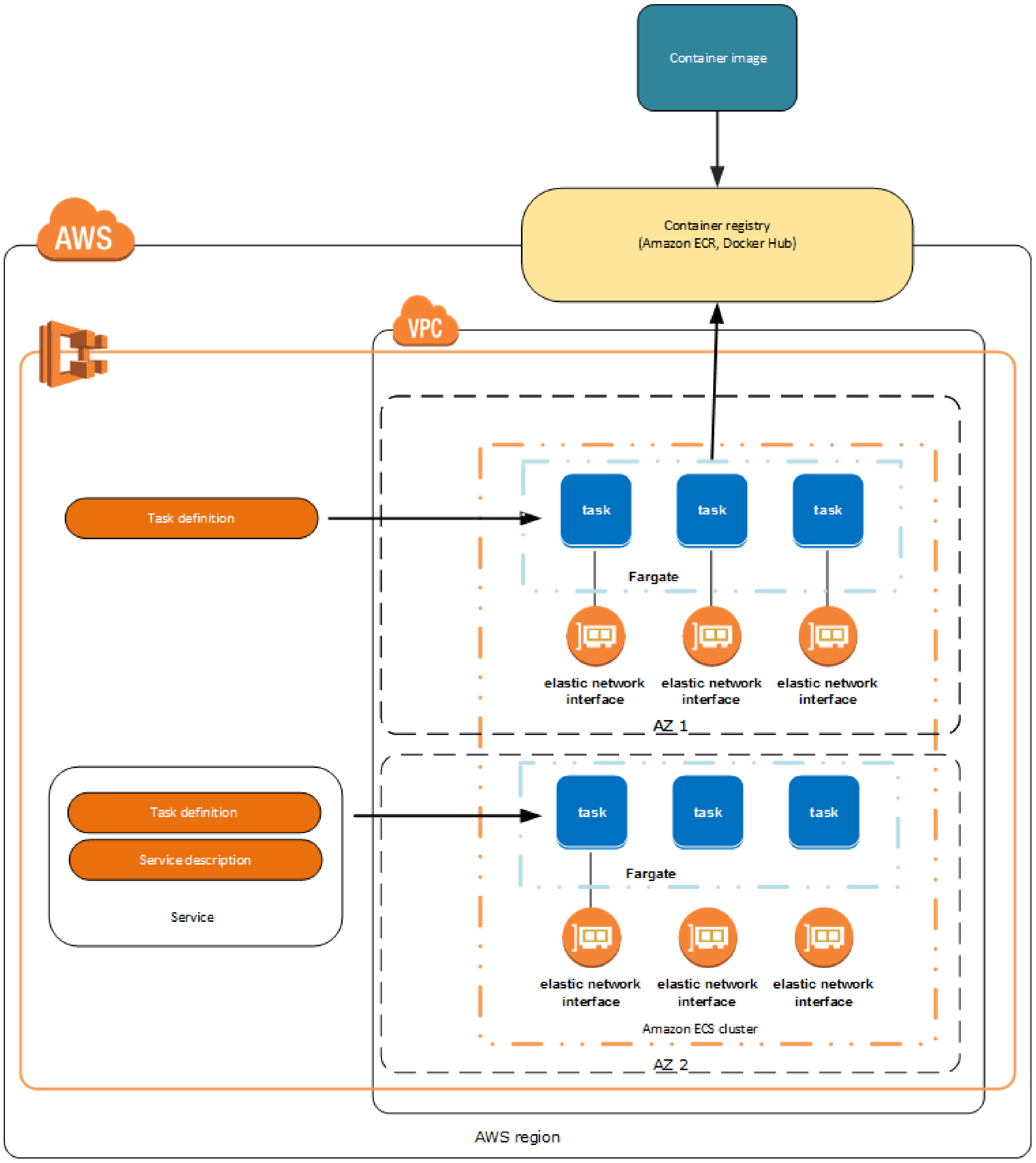
AWS ECS AND EKS (CLOUDFORMATION)

AWS ECS



Amazon Elastic Container Service (Amazon ECS) is a highly scalable, fast, container management service that makes it easy to run, stop, and manage Docker containers on a cluster.

Amazon ECS is a regional service that simplifies running application containers in a highly available manner across multiple Availability Zones within a Region. You can create Amazon ECS clusters within a new or existing VPC. After a cluster is up and running, you can define task definitions and services that specify which Docker container images to run across your clusters. Container images are stored in and pulled from container registries, which may exist within or outside of your AWS infrastructure.

To deploy applications on Amazon ECS, your application components must be architected to run in *containers*.

Task Definitions

To prepare your application to run on Amazon ECS, you create a *task definition*. The task definition is a text file, in JSON format, that describes one or more containers, up to a maximum of ten, that form your application. It can be thought of as a blueprint for your application.

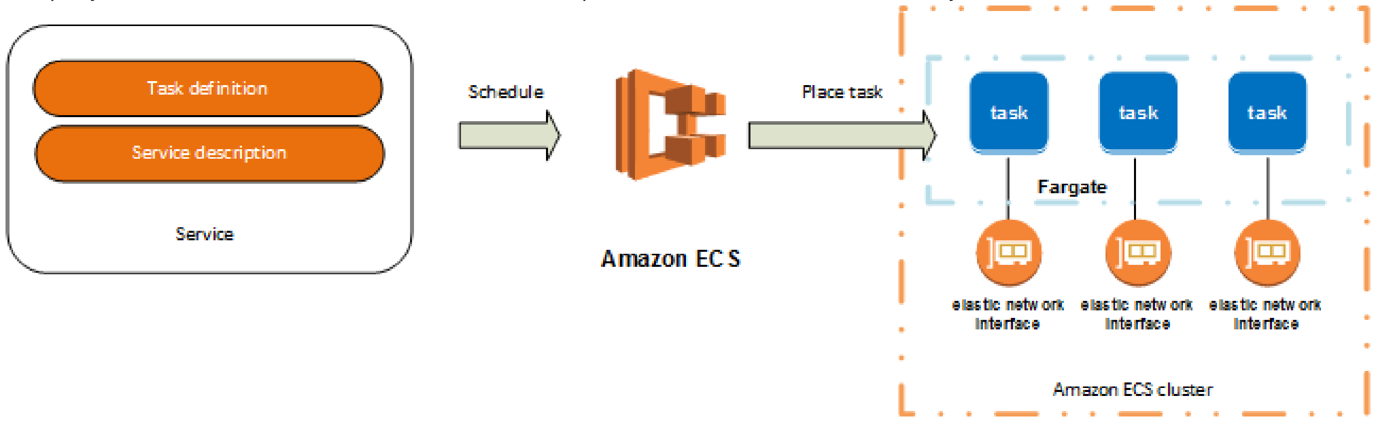
Example,

```
{
"family": "webserver",
"containerDefinitions":
     [
          {
               "name": "web",
               "image": "nginx",
               "memory": "100",
               "cpu": "99"
          },
     ],
"requiresCompatibilities": [ "FARGATE" ],
"networkMode": "awsvpc",
"memory": "512",
"cpu": "256",
}
```

Tasks and Scheduling

A *task* is the instantiation of a task definition within a cluster. After you have created a task definition for your application within Amazon ECS, you can specify the number of tasks that will run on your cluster.

The Amazon ECS task scheduler is responsible for placing tasks within your cluster. There are several different scheduling options available. For example, you can define a *service* that runs and maintains a specified number of tasks simultaneously.

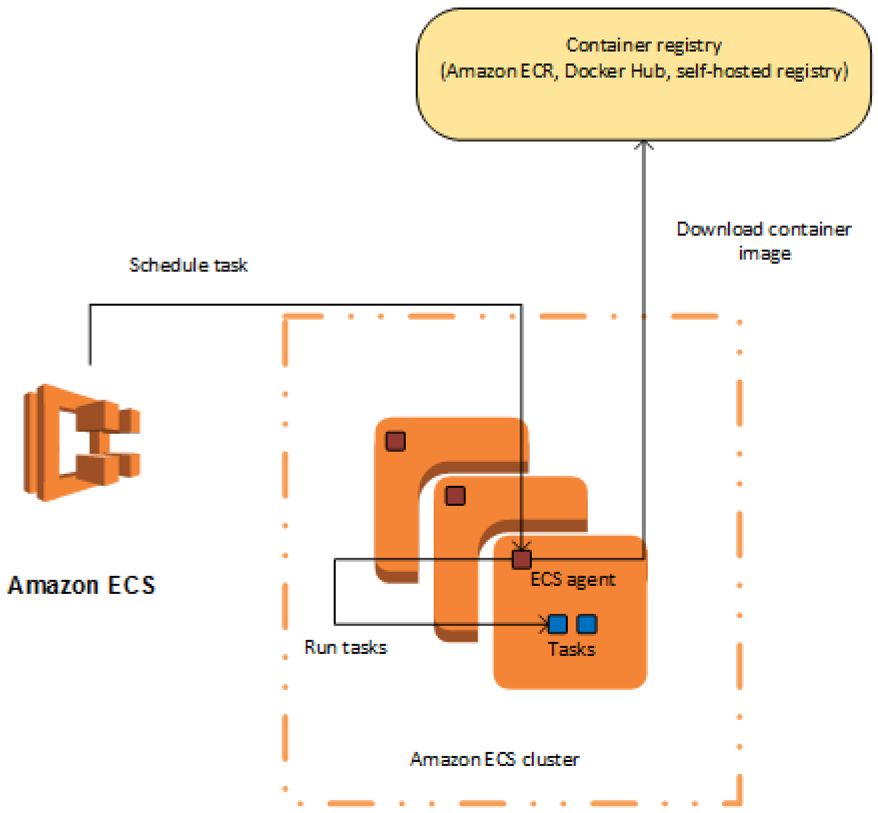


Clusters

When you run tasks using Amazon ECS, you place them on a *cluster*, which is a logical grouping of resources. When using the Fargate launch type with tasks within your cluster, Amazon ECS manages your cluster resources. When using the EC2 launch type, then your clusters are a group of container instances you manage. An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent. Amazon ECS downloads your container images from a registry that you specify, and runs those images within your cluster.

Container Agent

The *container agent* runs on each infrastructure resource within an Amazon ECS cluster. It sends information about the resource's current running tasks and resource utilization to Amazon ECS, and starts and stops tasks whenever it receives a request from Amazon ECS.



OVERVIEW

- ECS utilizes EC2 underneath to power its compute resources. The EC2 instances are deployed and managed within your AWS resources and registered to an ECS Cluster that you define.
- The ECS Cluster can be set up in an already existing VPC, taking advantage of the AWS infrastructure you may have in place, and manages the deployment of containers to the EC2 instances. The metadata specifying how a container is deployed and runs corresponds to an ECS Task.
- ECS Tasks detail the repository for the docker image, port mappings, environment variables, and volumes to attach. Most everything used in a command line run from Docker will be mapped to the Task definition.
- Once your Tasks are defined, ECS Services detail the auto-scaling nature of the Tasks. If a Task is stopped, the corresponding ECS Service can restart the Task or launch a new instance to replace it.

EC2 vs. FARGATE

- ECS has two launch types that can define how the compute resources will be managed. The traditional EC2 launch type detailed in the overview above utilizes your own EC2 instances. Fargate removes the responsibility of provisioning, configuring and managing the EC2 instances by allowing AWS to manage the EC2 instances.
- Fargate eliminates the need to manage servers, but also puts a requirement of your Task definitions to be stateless. Currently, no volumes can be attached to the containers defined in your Tasks which eliminates some types of containers (such as datastores requiring persistent storage) to be able to run in the ECS environment.

Why use it?

ECS EC2 (traditional)

- You are all-in on AWS
- You have an existing AWS footprint such as an already defined VPC and subnets, processes for provisioning and managing EC2 instances
- You have an Ops team in place to manage AWS resources

ECS Fargate

- You are mostly utilizing AWS and may or may not have an existing VPC
- You may or may not be currently utilizing ECS and do not want or are looking to reduce your operations responsibility on AWS
- Your applications are stateless or manage state through S3, RDS or other external services from your containers.

HOW TO DEPLOY ECS USING CLOUDFORMATION

- We will use awscli to deploy cloudformation template.

```
{
  "AWSTemplateFormatVersion":"2010-09-09",
  "Parameters":{
    "KeyName":{
      "Type":"AWS::EC2::KeyPair::KeyName",
   "Default":"python_app",
      "Description":"Name of an existing EC2 KeyPair to enable SSH
access to the ECS instances."
    },
    "VpcId":{
      "Type":"AWS::EC2::VPC::Id",
   "Default":"vpc-9050c3eb",
      "Description":"Select a VPC that allows instances to access the
Internet."
    },
    "SubnetId":{
      "Type":"List<AWS::EC2::Subnet::Id>",
   "Default": "subnet-36f3c919",
      "Description":"Select at two subnets in your selected VPC."
    },
    "DesiredCapacity":{
      "Type":"Number",
      "Default":"1",
      "Description":"Number of instances to launch in your ECS cluster."
    },
    "MaxSize":{
      "Type":"Number",
      "Default":"1",
      "Description":"Maximum number of instances that can be launched in
your ECS cluster."
    },
    "InstanceType":{
      "Description":"EC2 instance type",
      "Type":"String",
      "Default":"t2.micro",
      "ConstraintDescription":"Please choose a valid instance type."
    }
  },
  "Resources":{
    "ECSCluster":{
      "Type":"AWS::ECS::Cluster",
   "Properties" : {
      "ClusterName" : "Nova-Test"
   }

   },
    "taskdefinition":{
      "Type":"AWS::ECS::TaskDefinition",
      "Properties":{
          "Family": "Nova-Test",
          "ContainerDefinitions":[
            {
              "Name":"python-app",
              "Cpu":"10",
              "Essential":"true",

  "Image":"536285340728.dkr.ecr.us-east-1.amazonaws.com/add2vals:0.1.1-2",
              "Memory":"256"
              }

          ]
        }
      },
      "ECSAutoScalingGroup":{
        "Type": "AWS::AutoScaling::AutoScalingGroup",
        "Properties": {
          "AvailabilityZones": [
            "us-east-1c"
          ],
          "Cooldown": "300",
          "DesiredCapacity": "1",
          "HealthCheckGracePeriod": "0",
          "HealthCheckType": "EC2",
          "MaxSize": "1",
          "MinSize": "0",
          "VPCZoneIdentifier": [
            "subnet-36f3c919"
          ],
          "LaunchConfigurationName": {
            "Ref": "ContainerInstances"
          },
          "TerminationPolicies": [
            "Default"
          ]
        }
      },
      "ContainerInstances":{
        "Type": "AWS::AutoScaling::LaunchConfiguration",
        "Properties": {
          "AssociatePublicIpAddress": true,
          "ImageId": "ami-0b84afb18c43907ba",
          "InstanceType": "t2.micro",
          "KeyName": "python_app",
    "UserData":{
            "Fn::Base64":{
              "Fn::Join":[
                "",
                [
                  "#!/bin/bash -xe\n",
                  "echo ECS_CLUSTER=",
                  {
                    "Ref":"ECSCluster"
                  },
                  " >> /etc/ecs/ecs.config\n",
                  "yum install -y aws-cfn-bootstrap\n",
                  "/opt/aws/bin/cfn-signal -e $? ",
                  "         --stack ",
                  {
                    "Ref":"AWS::StackName"
                  },
                  "         --resource ECSAutoScalingGroup ",
                  "         --region ",
                  {
                    "Ref":"AWS::Region"
                  },
                  "\n"
                ]
              ]
            }
  },
        "IamInstanceProfile":
"arn:aws:iam::536285340728:instance-profile/ecsInstanceRole",
        "InstanceMonitoring": "true",
        "SecurityGroups": [
          "sg-0eb21836ffe9d043d"
        ],
        "BlockDeviceMappings": [
          {
            "DeviceName": "/dev/xvdcz",
            "Ebs": {
              "VolumeSize": 22
            }
          }
        ]
      }
    },
    "service":{
      "Type":"AWS::ECS::Service",
      "Properties":{
        "Cluster":{
          "Ref":"ECSCluster"
        },
        "DesiredCount":"1",
        "TaskDefinition":{
          "Ref":"taskdefinition"
        }
      }
    }
    }

  }
```

- This template will create a ECS cluster with name Nova-test. It is using EC2 as the launch type which is default.
- To deploy this template run the following command,

> *aws2 cloudformation deploy --template-file {file_name}.json --stack-name {name]*

- Now we will see the CFN template for ECS with FARGATE

```
  {
    "AWSTemplateFormatVersion":"2010-09-09",
    "Parameters":{
      "KeyName":{
        "Type":"AWS::EC2::KeyPair::KeyName",
     "Default":"python_app",
        "Description":"Name of an existing EC2 KeyPair to enable SSH
  access to the ECS instances."
      },
      "VpcId":{
        "Type":"AWS::EC2::VPC::Id",
     "Default":"vpc-9050c3eb",
        "Description":"Select a VPC that allows instances to access the
  Internet."
      },
      "SubnetId":{
        "Type":"List<AWS::EC2::Subnet::Id>",
     "Default": "subnet-36f3c919",
        "Description":"Select at two subnets in your selected VPC."
      },
      "DesiredCapacity":{
        "Type":"Number",
        "Default":"1",
        "Description":"Number of instances to launch in your ECS cluster."
      },
      "MaxSize":{
        "Type":"Number",
        "Default":"1",
        "Description":"Maximum number of instances that can be launched in
  your ECS cluster."
      },
      "InstanceType":{
        "Description":"EC2 instance type",
        "Type":"String",
        "Default":"t2.micro",
        "ConstraintDescription":"Please choose a valid instance type."
      }
    },
    "Resources":{
```

```
    "ECSCluster":{
      "Type":"AWS::ECS::Cluster",
   "Properties" : {
      "ClusterName" : "Nova-Test"
   }

   },
    "taskdefinition":{
      "Type":"AWS::ECS::TaskDefinition",
      "Properties":{
        "Family": "Nova-Test",
  "RequiresCompatibilities" : ["EC2", "FARGATE"],
  "ExecutionRoleArn" : "arn:aws:iam::536285340728:role/nova_ecs",
  "NetworkMode" : "awsvpc",
  "Cpu" : "256",
  "Memory" : "512",
        "ContainerDefinitions":[
          {
            "Name":"python-app",
            "Essential":"true",

"Image":"536285340728.dkr.ecr.us-east-1.amazonaws.com/add2vals:0.1.1-2"
            }

        ]
      }
    },
    "ECSAutoScalingGroup":{
      "Type": "AWS::AutoScaling::AutoScalingGroup",
      "Properties": {
        "AvailabilityZones": [
          "us-east-1c"
        ],
        "Cooldown": "300",
        "DesiredCapacity": "1",
        "HealthCheckGracePeriod": "0",
        "HealthCheckType": "EC2",
        "MaxSize": "1",
        "MinSize": "0",
        "VPCZoneIdentifier": [
          "subnet-36f3c919"
        ],
        "LaunchConfigurationName": {
          "Ref": "ContainerInstances"
        },
        "TerminationPolicies": [
          "Default"
        ]
      }
    },
      "ContainerInstances":{
        "Type": "AWS::AutoScaling::LaunchConfiguration",
        "Properties": {
          "AssociatePublicIpAddress": true,
          "ImageId": "ami-0b84afb18c43907ba",
          "InstanceType": "t2.micro",
          "KeyName": "python_app",
    "UserData":{
            "Fn::Base64":{
              "Fn::Join":[
                "",
                [
                  "#!/bin/bash -xe\n",
                  "echo ECS_CLUSTER=",
                  {
                    "Ref":"ECSCluster"
                  },
                  " >> /etc/ecs/ecs.config\n",
                  "yum install -y aws-cfn-bootstrap\n",
                  "/opt/aws/bin/cfn-signal -e $? ",
                  "         --stack ",
                  {
                    "Ref":"AWS::StackName"
                  },
                  "         --resource ECSAutoScalingGroup ",
                  "         --region ",
                  {
                    "Ref":"AWS::Region"
                  },
                  "\n"
                ]
              ]
            }
    },
          "IamInstanceProfile":
  "arn:aws:iam::536285340728:instance-profile/ecsInstanceRole",
          "InstanceMonitoring": "true",
          "SecurityGroups": [
            "sg-0eb21836ffe9d043d"
          ],
          "BlockDeviceMappings": [
            {
              "DeviceName": "/dev/xvdcz",
              "Ebs": {
                "VolumeSize": 22
              }
            }
          ]
        }
      },
    "service":{
      "Type":"AWS::ECS::Service",
      "Properties":{
        "Cluster":{
          "Ref":"ECSCluster"
        },
  "NetworkConfiguration": {
   "AwsvpcConfiguration": {
   "AssignPublicIp": "ENABLED",
   "SecurityGroups": [ "sg-0eb21836ffe9d043d" ],
   "Subnets": [ "subnet-36f3c919" ]
  }
 },
        "DesiredCount":"1",
  "LaunchType" : "FARGATE",
        "TaskDefinition":{
          "Ref":"taskdefinition"
        }
      }
    }
    }

  }
```

- There are some parameters whose values are defined on the basis of Launch Type we select.
- To deploy this template run the following command,

> *aws2 cloudformation deploy --template-file {file_name}.json --stack-name {name}*

AWS EKS

EKS is the managed Kubernetes hosting environment supported by AWS. If you are already using an on-premise or cloud provided host for Kubernetes, or are looking to utilize a defacto industry standard for open source orchestration of containers, EKS can provide many benefits of Kubernetes without the operations responsibility of hosting and configuring the Kubernetes environment.

Overview

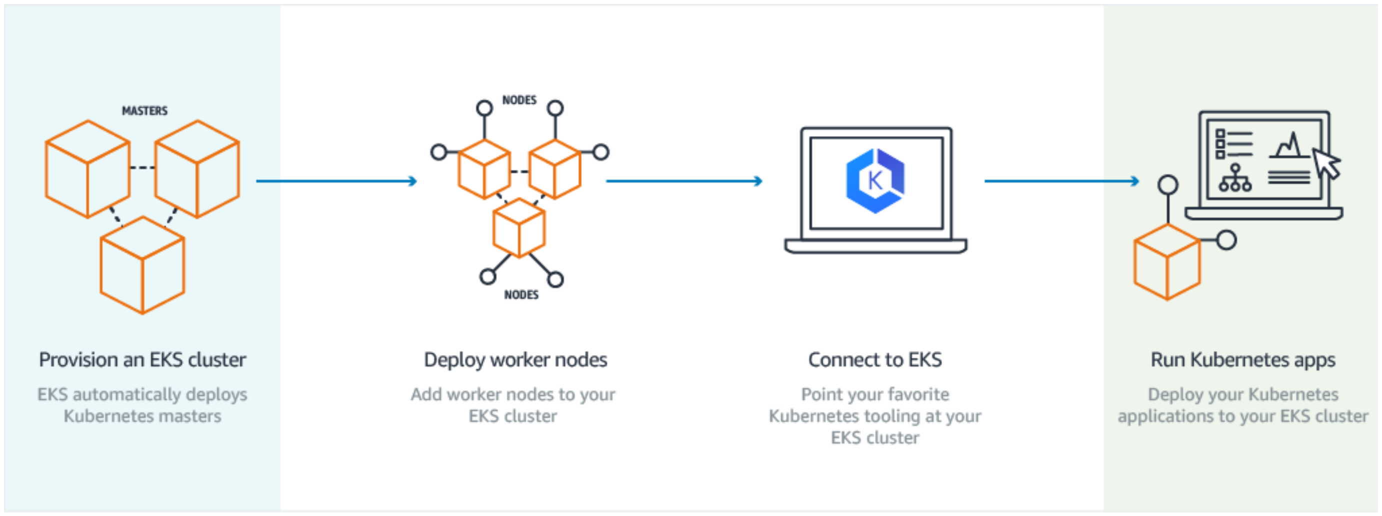
EKS utilizes EC2 instances that are specifically defined by AWS to be Kubernetes Nodes. The Nodes are registered with an AWS managed Kubernetes Control Plane which is all defined within your EKS Cluster.

Once your EKS Cluster is configured, the management, monitoring, and deployment of infrastructure will move away from the custom AWS specific processes to the open source Kubernetes processes. For example, a majority of the configuration of your cluster and monitoring of your pods will be handled by the Kubernetes command line, kubectl.

Why use it?

- You are utilizing AWS but want to keep your infrastructure portable for other cloud providers
- You want to utilize the most flexible and future-proof managed container environment
- You want to ensure open source tools are used as much as possible
- You want developers and operations to be able to determine how containers are managed

How Does Amazon EKS Work?


MASTERS
NODES
NODES
Provision an EKS cluster
EKS automatically deploys Kubernetes masters
Deploy worker nodes
Add worker nodes to your EKS cluster
Connect to EKS
Point your favorite Kubernetes tooling at your EKS cluster
Run Kubernetes apps
Deploy your Kubernetes applications to your EKS cluster

HOW TO DEPLOY EKS USING CLOUDFORMATION

- We will be having a Json template that will create an EKS cluster and add a Node to it, but to deploy apps on it we will need to install some tools to interact with Kubernetes.
- We will use kubectl to interact with the kubernetes cluster.
- The CFN template is as follows,

```
{
  "AWSTemplateFormatVersion":"2010-09-09",
  "Resources":{
 "myCluster": {
            "Type": "AWS::EKS::Cluster",
            "Properties": {
                "Name": "Nova_test",
                "RoleArn": "arn:aws:iam::536285340728:role/nova_eks",
                "ResourcesVpcConfig": {
                    "SecurityGroupIds": [
                        "sg-0eb21836ffe9d043d"
                    ],
                "SubnetIds": [
                    "subnet-5a615a07",
                    "subnet-36f3c919 "
                ]
            }
  }
    },
 "EKSNodegroup": {
            "Type": "AWS::EKS::Nodegroup",
            "Properties": {
    "AmiType": "AL2_x86_64",
                "ClusterName": {"Ref":"myCluster"},
                "NodeRole":
"arn:aws:iam::536285340728:role/novaEKSrole-NodeInstanceRole-OJD4W20HTXF
F",
                "ScalingConfig": {
                    "MinSize": 1,
                    "DesiredSize": 1,
                    "MaxSize": 1
                },
                "Subnets": [
                    "subnet-5a615a07",
                    "subnet-36f3c919 "
                ]
            }
        }
 }
}
```

- To deploy this template use the following command,

> *aws2 cloudformation deploy --template-file {file_name}.json --stack-name {name}*

- Run the follwing command to create the config file,

> *aws2 eks --region us-east-1 update-kubeconfig --name {Cluster Name}*

- To verify that your cluster is availabe, run the following command,

> *kubectl get svc*

you will get the following output,

> *NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE*
> *kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 36m*

- To deploy an application we need a pod/deployment/service file, whose content are as follows

```
---
apiVersion: v1
kind: Pod
metadata:
  name: add2vals
spec:
  containers:
    - name: add2vals
      image:
536285340728.dkr.ecr.us-east-1.amazonaws.com/add2vals:0.1.1-2
      command: [ "/bin/bash", "-c", "--" ]
      args: [ "while true; do sleep 30; done;" ]
```

- To deploy this yaml, run the following command

> *kubectl apply -f test_deploy.yaml*

- To verify that your pod got created run the following command,

> *kubectl get pod*

- To run your container, use the following command

> *kubectl exec -it {pod_name} -- /bin/bash*

Jenkins Slave Setup In ECS

1. Go to `Manage Jenkins` –> `Manage Plugins` and search for `Amazon EC2 Container Service Plugin`
2. Install the plugin and restart it.
3. Add AWS access key and secret key to the Jenkins credentials.

Follow the steps given below for configuring the ECS plugin to integrate the ECS cluster.

1. Go to `Manage Jenkins` –> `Configure System` and search for cloud option.
2. Click `Add a new cloud` dropdown and select `Amazon EC2 Container Service Cloud`
3. You need to fill up the following details.

   Name: User defined name.

   Amazon ECS Credentials: If you are using AWS access keys, select the relevant credential. If you using AWS role, leave it empty. The cluster will get listed automatically.

   Amazon ECS Region Name: AWS region where you have the cluster.

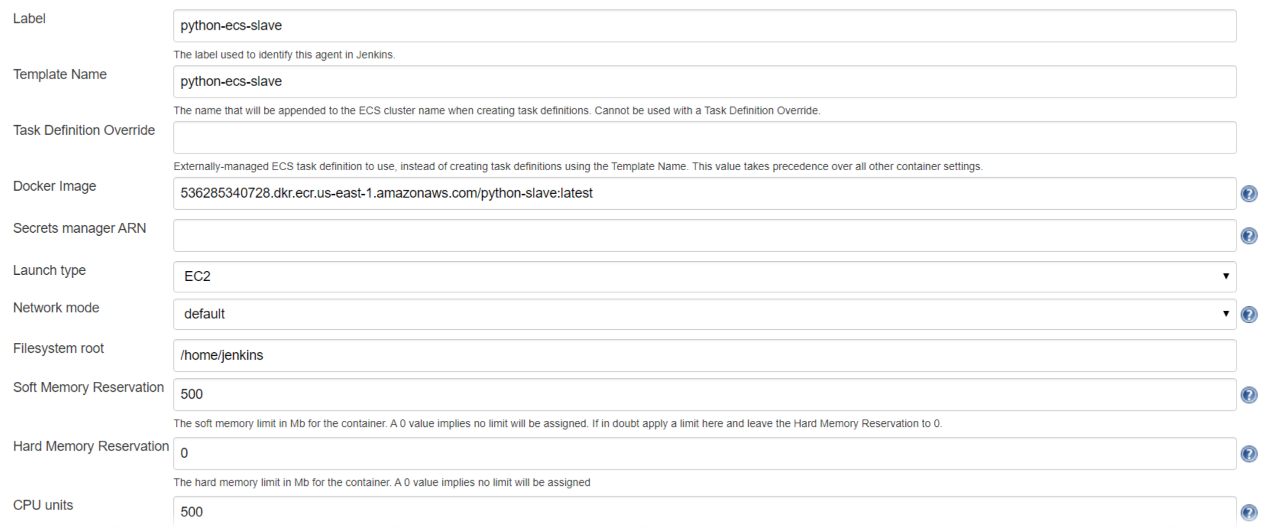
   ECS Cluster: Select your ECS cluster from the drop-down.

   An example is shown below.

4. If you are running jenkins master under an ELB, you need to add the `tunnel configuration` in the advanced section.

   The `Tunnel connection through` option should have the elb URL followed by the JNLP port as shown above.
5. Next, you need to add a `slave template` with a docker image which acts as a slave node. To do this, under `ECS slave templates` in the cloud configuration, click `add` and enter the details.
6. In the following example, I have added a ecr image with label python-ecs-salve. Label is very important, as we use the same name in our

JenkinsFile.



Label	python-ecs-slave
	The label used to identify this agent in Jenkins.
Template Name	python-ecs-slave
	The name that will be appended to the ECS cluster name when creating task definitions. Cannot be used with a Task Definition Override.
Task Definition Override	
	Externally-managed ECS task definition to use, instead of creating task definitions using the Template Name. This value takes precedence over all other container settings.
Docker Image	536285340728.dkr.ecr.us-east-1.amazonaws.com/python-slave:latest
Secrets manager ARN	
Launch type	EC2
Network mode	default
Filesystem root	/home/jenkins
Soft Memory Reservation	500
	The soft memory limit in Mb for the container. A 0 value implies no limit will be assigned. If in doubt apply a limit here and leave the Hard Memory Reservation to 0.
Hard Memory Reservation	0
	The hard memory limit in Mb for the container. A 0 value implies no limit will be assigned
CPU units	500

7. Dockerfile for Jenkins-Slave,

```
FROM jenkins/slave:3.35-3
ARG user=jenkins
USER root
RUN apt-get update && \
   apt-get install --no-install-recommends -y \
   python-pip \
   python-dev \
   pylint \
   binutils \
   && pip install --upgrade setuptools \
   && pip install wheel \
   && pip install xmlrunner \
   && pip install pyinstaller \
   && pip install nose coverage nosexcover \
   && rm -rf /tmp/downloaded_packages/ /tmp/*.rds \
   && rm -rf /var/lib/apt/lists/*
COPY jenkins-agent /usr/local/bin/jenkins-agent
RUN chmod +x /usr/local/bin/jenkins-agent &&\
    ln -s /usr/local/bin/jenkins-agent /usr/local/bin/jenkins-slave
USER ${user}
ENTRYPOINT ["jenkins-agent"]
```

Note : We need jenkins-agent script to make this Docker Image work as a jenkins agent.

jenkins-agent