AWS® Certified Cloud Practitioner

STUDY GUIDE

FOUNDATIONAL (CLF-C01) EXAM

Includes interactive online learning environment and study tools:

Two custom practice exams

100 electronic flashcards

Searchable key term glossary

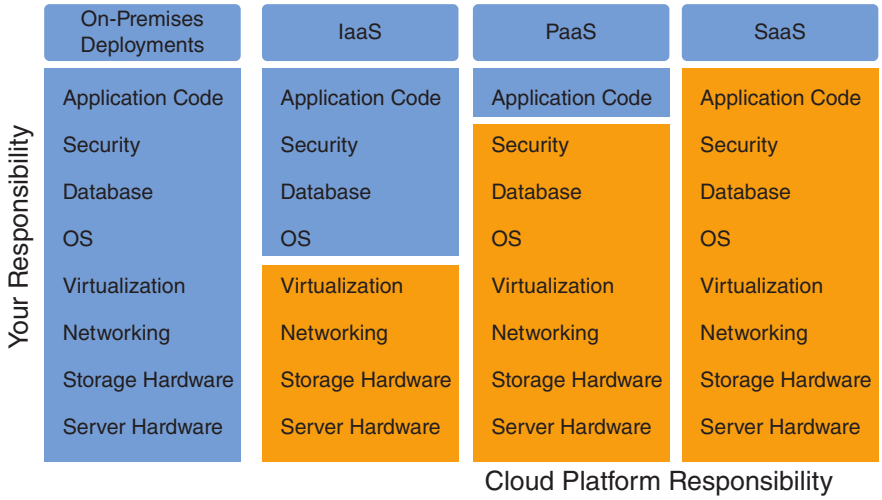
BEN PIPER
DAVID CLINTON

SYBEX
A Wiley Brand

While some may disagree with the designation, AWS SaaS products arguably include Simple Email Service and Amazon WorkSpaces.

Figure 1.2 compares the scope of responsibility you have on IaaS, PaaS, and SaaS platforms with the way it works for on-premises deployments.

FIGURE 1.2 The breakdown of responsibility across multiple infrastructure types



Serverless Workloads

Besides doing an excellent job emulating traditional server behavior, cloud providers can also enable entirely new ways to administrate applications and data. Perhaps the most obvious example is serverless computing.

Now don't be fooled by the name. You can't run a compute function without a computer environment (a "server") somewhere that'll host it. What "serverless" does allow is for individual developers to run their code for seconds or minutes at a time on some else's cloud servers.

The serverless model—as provided by services like AWS Lambda—makes it possible to design code that *reacts* to external events. When, for instance, a video file is uploaded to a repository (like an AWS S3 bucket or even an on-premises FTP site), it can trigger a Lambda function that will convert the file to a new video format. There's no need to maintain and pay for an actual instance running 24/7, just for the moments your code is actually running. And there's no administration overhead to worry about.

according to demand. Should, say, your WordPress site go viral and attract millions of viewers one day, AWS will invisibly ramp up the infrastructure to meet demand. As demand falls, your infrastructure will similarly drop. Keep this in mind, as such variations in demand will determine how much you'll be billed each month.

Deploying Container and Serverless Workloads

Even virtualized servers like EC2 instances tend to be resource-hungry. They do, after all, act like discrete, stand-alone machines running on top of a full-stack operating system. That means that having 5 or 10 of those virtual servers on a single physical host involves some serious duplication because each one will require its own OS kernel and device drivers.

Containers

Container technologies such as Docker avoid a lot of that overhead by allowing individual containers to share the Linux kernel with the physical host. They're also able to share common elements (called *layers*) with other containers running on a single host. This makes Docker containers fast to load and execute and also lets you pack many more container workloads on a single hardware platform.

You're always free to fire up one or more EC2 instances, install Docker, and use them to run as many containers as you'd like. But keeping all the bits and pieces talking to each other can get complicated. Instead, you can use either *Amazon Elastic Container Service* (ECS) or *Amazon Elastic Container Service for Kubernetes* (EKS) to orchestrate swarms of Docker containers on AWS using EC2 resources. Both of those services manage the underlying infrastructure for you, allowing you to ignore the messy details and concentrate on administrating Docker itself.

What's the difference between ECS and EKS? Broadly speaking, they both have the same big-picture goals. But EKS gets there by using the popular open source Kubernetes orchestration tool. They are different paths to the same place.

Serverless Functions

The serverless computing model uses a resource footprint that's even smaller than the one left by containers. Not only do *serverless functions* not require their own OS kernel, but they tend to spring into existence, perform some task, and then just as quickly die within minutes, if not seconds.

On the surface, Amazon's serverless service—AWS Lambda—looks a bit like Elastic Beanstalk. You define your function by setting a runtime environment (like Node.js, .NET, or Python) and uploading the code you want the function to run. But, unlike Beanstalk,

Lambda functions run only when triggered by a preset event. It could be a call from your mobile application, a change to a separate AWS resources (like an S3 bucket), or a log-based alert.

If an hour or a week passes without a trigger, Lambda won't launch a function (and you won't be billed anything). If there are a thousand concurrent executions, Lambda will scale automatically to meet the demand. Lambda functions are short-lived: they'll time out after 15 minutes.

Summary

Configuring EC2 instances is designed to mirror the process of provisioning and launching on-premises servers. Instances are defined by your choice of AMIs, instance type, storage volumes, and pricing model.

AMIs are organized into four categories: Quick Start, custom (My AMIs), AWS Marketplace, and Community. You can create your own AMI from a snapshot based on the EBS volume of an EC2 instance.

EC2 instance types are designed to fit specific application demands, and individual optimizations are generally available in varying sizes.

EBS storage volumes can be encrypted and are more like physical hard drives in the flexibility of their usage. Instance store volumes are located on the same physical server hosting your instance and will, therefore, deliver faster performance.

EC2 on-demand pricing is best for short-term workloads that can't be interrupted. Longer-term workloads—like ecommerce websites—will often be much less expensive when purchased as reserved instances. Spot instances work well for compute-intensive data operations that can survive unexpected shutdowns.

Lightsail, Elastic Beanstalk, Elastic Container Service, Elastic Container Service for Kubernetes, and Lambda are all designed to provide abstracted compute services that simplify, automate, and reduce the cost of compute operations.

Exam Essentials

Understand the elements required to provision an EC2 instance. An instance requires a base OS (AMI) and—optionally—an application stack, an instance type for its hardware profile, and either an EBS or an instance volume for storage.

Understand the sources, pricing, and availability of EC2 AMIs. The Quick Start and Marketplace AMIs are supported by Amazon or a recognized third-party vendor, which may not be true of AMIs selected from the Community collection. In any case, you should confirm whether using a particular AMI will incur extra charges beyond the normal EC2 usage.

AWS®

Certified Developer

Official Study Guide

Associate (DVA-C01) Exam

AWS CodePipeline provides a number of built-in integrations to other AWS services, such as AWS CloudFormation, AWS CodeBuild, AWS CodeCommit, AWS CodeDeploy, Amazon Elastic Container Service (ECS), Elastic Beanstalk, AWS Lambda, AWS OpsWorks Stacks, and Amazon Simple Storage Service (Amazon S3). Some partner tools include GitHub (`https://github.com`) and Jenkins (`https://jenkins.io`). Customers also have the ability to create their own integrations, which provides a great degree of flexibility.

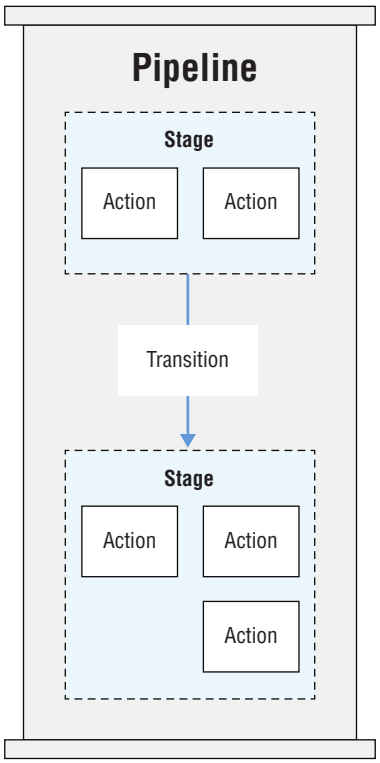
You define workflow steps through a visual editor within the AWS Management Console or via a JavaScript Object Notation (JSON) structure for use in the AWS CLI or AWS SDKs. Access to create and manage release workflows is controlled by AWS Identity and Access Management (IAM). You can grant users fine-grained permissions, controlling what actions they can perform and on which workflows.

AWS CodePipeline provides a dashboard where you can review real-time progress of revisions, attempt to retry failed actions, and review version information about revisions that pass through the pipeline.

AWS CodePipeline Concepts

There are a number of different components that make up AWS CodePipeline and the workflows (*pipelines*) created by customers. Figure 7.3 displays the AWS CodePipeline concepts.

FIGURE 7.3 Pipeline structure



Pipeline

A *pipeline* is the overall workflow that defines what transformations software changes will undergo.

You cannot change the name of a pipeline. If you would like to change the name, you must create a new pipeline.

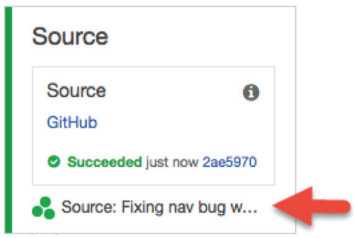
Revision

A *revision* is the work item that passes through a pipeline. It can be a change to your source code or data stored in AWS CodeCommit or GitHub or a change to the version of an archive in Amazon S3. A pipeline can have multiple revisions flowing through it at the same time, but a single stage can process one revision at a time. A revision is immediately picked up by a source action when a change is detected in the source itself (such as a commit to an AWS CodeCommit repository).

If you use Amazon S3 as a source action, you must enable versioning on the bucket.

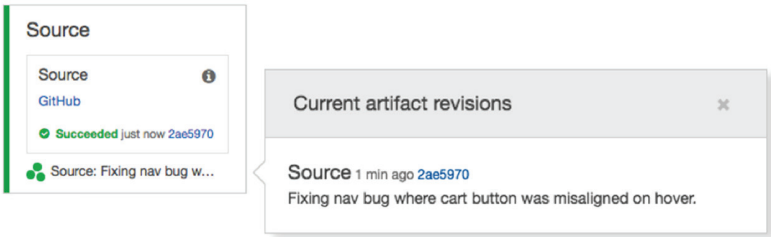
Details of the most recent revision to pass through a stage are kept within the stage itself and are accessible from the console or AWS CLI. To see the last revision that was passed through a source stage, for example, you can select the revision details at the bottom of the stage, as shown in Figure 7.4.

FIGURE 7.4 Source stage



Depending on the source type (Amazon S3, AWS CodeCommit, or GitHub), additional information will be accessible from the revision details pane (such as a link to the commit on `https://github.com`), as shown in Figure 7.5.

FIGURE 7.5 Revision details



Stage

A *stage* is a group of one or more actions. Each stage must have a unique name. Should any one action in a stage fail, the entire stage fails for this revision.

Action

An *action* defines the work to perform on the revision. You can configure pipeline actions to run in series or in parallel. If all actions in a stage complete successfully for a revision, it passes to the next stage in the pipeline. However, if one action fails in the stage, the revision will not pass further through the pipeline. At this point, the stage that contains the failed action can be retried for the same revision. Otherwise, a new revision is able to pass through the stage.

A pipeline must have two or more stages. The first stage includes one or more source actions only. Only the first stage may include source actions.

Every action in the same stage must have a unique name.

Source

The *source* action defines the location where you store and update source files. Modifications to files in a source repository or archive trigger deployments to a pipeline. AWS CodePipeline supports these sources for your pipeline:

- Amazon S3
- AWS CodeCommit
- GitHub

A single pipeline can contain multiple source actions. If a change is detected in one of the sources, all source actions will be invoked.

To use GitHub as a source provider for AWS CodePipeline, you must authenticate to GitHub when you create a pipeline. You provide GitHub credentials to authorize AWS CodePipeline to connect to GitHub to list and view repositories accessible by the authenticating account. For this link, AWS recommends that you create a service account user so that the lifecycle of personal accounts is not tied to the link between AWS CodePipeline and GitHub.

After you authenticate GitHub, a link is created between AWS CodePipeline for this AWS region and GitHub. This allows IAM users to list repositories and branches accessible by the authenticated GitHub user.

Build

You use a *build* action to define tasks such as compiling source code, running unit tests, and performing other tasks that produce output artifacts for later use in your pipeline. For example, you can use a build stage to import large assets that are not part of a source bundle into the artifact to deploy it to Amazon Elastic Compute Cloud (Amazon EC2) instances. AWS CodePipeline supports the integrations for the following build actions:

- AWS CodeBuild
- CloudBees
- Jenkins
- Solano CI
- TeamCity

Test

You can use *test* actions to run various tests against source and compiled code, such as `lint` or syntax tests on source code, and unit tests on compiled, running applications. AWS CodePipeline supports the following test integrations:

- AWS CodeBuild
- BlazeMeter
- Ghost Inspector
- Hewlett Packard Enterprise (HPE) StormRunner Load
- Nouvola
- Runscope

Deploy

The *deploy* action is responsible for taking compiled or prepared assets and installing them on instances, on-premises servers, serverless functions, or deploying and updating infrastructure using AWS CloudFormation templates. The following services are supported as deploy actions:

- AWS CloudFormation
- AWS CodeDeploy
- Amazon Elastic Container Service
- AWS Elastic Beanstalk
- OpsWorks Stacks
- Xebia Labs

Approval

An *approval* action is a manual gate that controls whether a revision can proceed to the next stage in a pipeline. Further progress by a revision is halted until a manual approval by an IAM user or IAM role occurs.

Specifically, the `codepipeline:PutApprovalResult` action must be included in the IAM policy.

Upon approval, AWS CodePipeline approves the revision to proceed to the next stage in the pipeline. However, if the revision is not approved (rejected or the approval expires), the change halts and will stop progress through the pipeline. The purpose of this action is to allow manual review of the code or other quality assurance tasks prior to moving further down the pipeline.

Approval actions cannot occur within source stages.

You must approve actions manually within seven days; otherwise, AWS CodePipeline rejects the code. When an approval action rejects, the outcome is equivalent to when the stage fails. You can retire the action, which initiates the approval process again. Approval actions provide several options that you can use to provide additional information about what you choose to approve.

Publish approval notifications Amazon Simple Notification Service (Amazon SNS) sends notices to one or more targets that approval is pending.

Rolling Deployments

You can issue commands to subsets of instances in a stack or layer at a time. If you split the deployment into multiple phases, the blast radius of failures will be minimized to only a few instances that you can replace, roll back, or repair.

Blue/Green Deployments (Separate Stacks)

Much like you use separate stacks for different environments of the same application, you can also use separate stacks for different deployments. This ensures that all features and updates to an application can be thoroughly tested before routing requests to the new environment. Additionally, you can leave the previous environment running for some time to perform backups, investigate logs, or perform other tasks.

When you use Elastic Load Balancing layers and Amazon Route 53, you can route traffic to the new environment with built-in weighted routing policies. You can progressively increase traffic to the new stack as health checks and other monitoring indicate the new application version has deployed without error.

Manage Databases Between Deployments

In either deployment strategy, there will likely be a backend database with which instances running either version will need to communicate. Currently, Amazon RDS layers support registering a database with only one stack at a time.

If you do not want to create a new database and migrate data as part of the deployment process, you can configure both application version instances to connect to the same database (if there are no schema changes that would prevent this). Whichever stack does not have the Amazon RDS instance registered will need to obtain credentials via another means, such as custom JSON or a configuration file in a secure Amazon S3 bucket.

If there are schema changes that are not backward compatible, create a new database to provide the most seamless transition. However, it will be important to ensure that data is not lost or corrupted during the transition process. You should heavily test this before you attempt it in a production deployment.

Using Amazon Elastic Container Service to Deploy Containers

Amazon ECS is a highly scalable, high-performance container orchestration service that supports Docker containers and allows you to easily run and scale containerized applications on AWS. Amazon ECS eliminates the need for you to install and operate your own container orchestration software, manage and scale a cluster of virtual machines, or schedule containers on those virtual machines.

With simple API calls, you can launch and stop Docker-enabled applications, query the complete state of your application, and access many familiar features such as IAM roles, security groups, load balancers, Amazon CloudWatch Events, AWS CloudFormation templates, and AWS CloudTrail logs.

What Is Amazon ECS?

Amazon ECS streamlines the process for managing and scheduling containers across fleets of Amazon EC2 instances, without the need to include separate management tools for container orchestration or cluster scaling. *AWS Fargate* reduces management further as it deploys containers to serverless architecture and removes cluster management requirements entirely. To create a cluster and deploy services, you need only configure the resource requirements of containers and availability requirements. Amazon ECS manages the rest with the use of an agent that runs on cluster instances. AWS Fargate requires no agent management.

To react to changes in demands for your service or application, Amazon ECS supports Amazon EC2 Auto Scaling groups of cluster instances that allow your service to increase running container counts across multiple instances as demand increases. You can define container isolation and dependencies as part of the service definition. You can use the service definition to enforce requirements without user interaction, such as "only one container of type A may run on a cluster instance at a time."

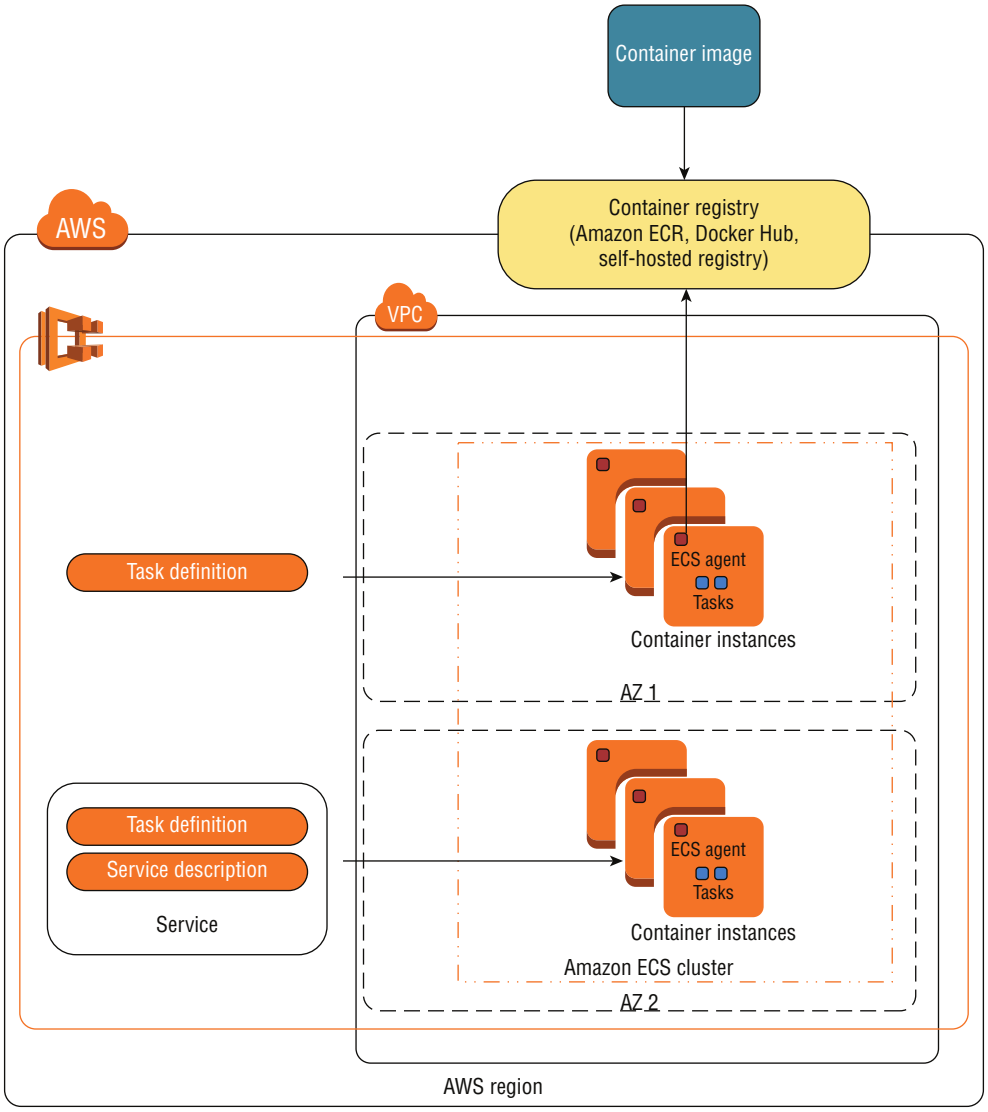
Amazon ECS Concepts

This section details Amazon ECS concepts.

Amazon ECS Cluster

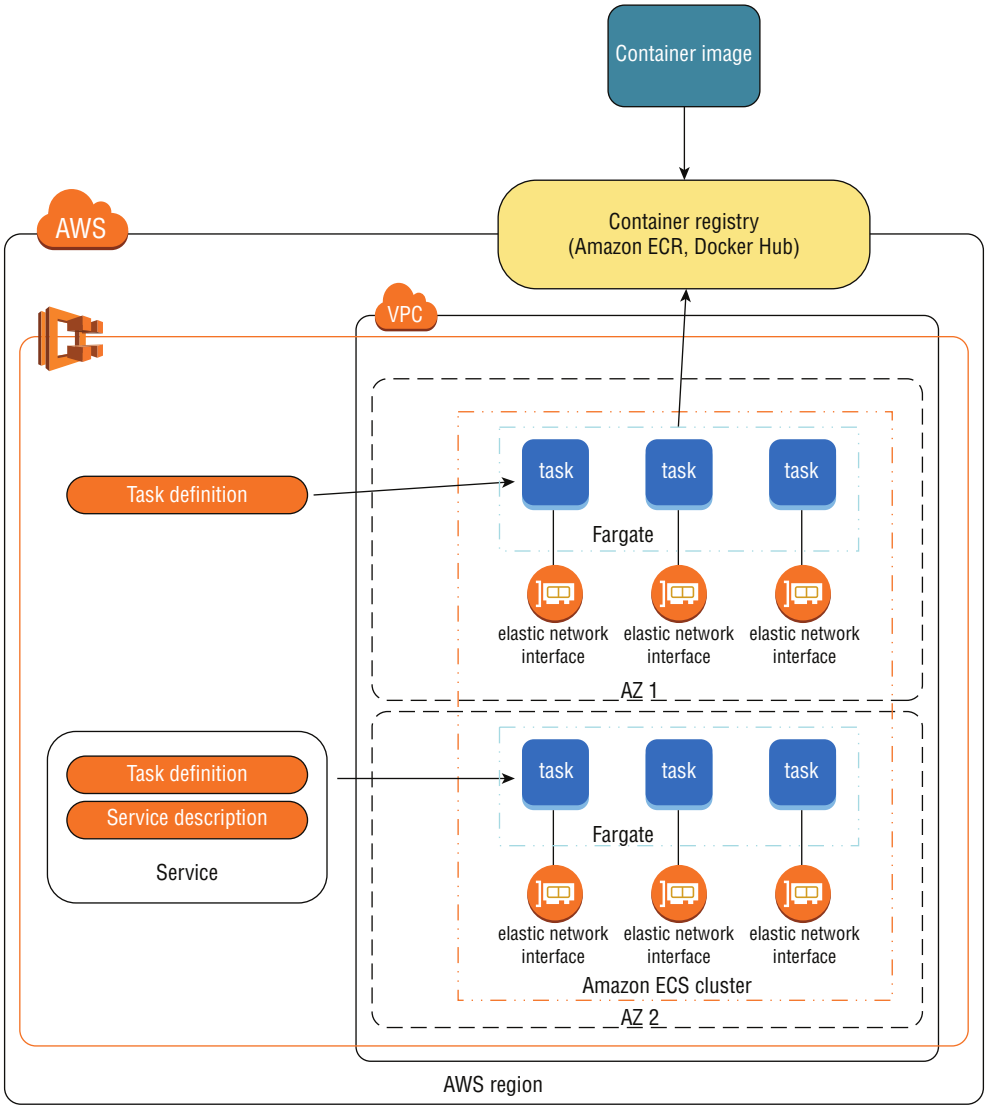
Amazon ECS clusters are the foundational infrastructure components on which containers run. Clusters consist of one or more Amazon EC2 instances in your Amazon VPC. Each instance in a cluster (cluster instance) has an agent installed. The agent is responsible for receiving container scheduling/shutdown commands from the Amazon ECS service and to report the current health status of containers (restart or replace). Figure 9.14 demonstrates an Amazon EC2 launch type, where instances make up the Amazon ECS cluster.

FIGURE 9.14 Amazon ECS architecture



In an AWS Fargate launch type, Amazon ECS clusters are no longer made up of Amazon EC2 instances. Since the tasks themselves launch on the AWS infrastructure, AWS assigns each one an elastic network interface with an Amazon VPC. This provides network connectivity for the container without the need to manage the infrastructure on which it runs. Figure 9.15 demonstrates an AWS Fargate cluster that runs in multiple availability zones (AZs).

FIGURE 9.15 AWS Fargate architecture



An individual cluster can support both Amazon EC2 and AWS Fargate launch types. However, a single cluster instance can belong to only one cluster at a time. Amazon EC2 launch types support both on-demand and spot instances, and they allow you to reduce cost for noncritical workloads.

To enable network connectivity for containers that run on your instance, the corresponding task definition must outline port mappings from the container to the host

instance. When you create a container instance, you can select the instance type to use. The compute resources available to this instance type will determine how many containers can be run on the instance. For example, if a `t2.micro` instance has one vCPU and 1 GB of RAM, it will not be able to run containers that require two vCPUs.

After you add a container instance to a cluster and you place containers on it, there may be situations where you would need to remove the container from the cluster temporarily—for a regular patch, for example. However, if critical tasks run on a container instance, you may want to wait for the containers to terminate gracefully. Container instance draining can be used to drain running containers from an instance and prevent new ones from being started. Depending on the service's configuration, replacement tasks start before or after the original tasks terminate.

- If the value of `minimumHealthyPercent` is less than 100 percent, the service will terminate the task and launch a replacement.
- If the value is greater than 100 percent, the service will attempt to launch a replacement task before it terminates the original.

To make room for launching additional tasks, you can scale out a cluster with Amazon EC2 Auto Scaling groups. For an EC2 Auto Scaling group to work with an Amazon ECS cluster, you must install the Amazon ECS agent either as part of the AMI or via instance `userdata`. To change the number container instances that run, you can adjust the size of the corresponding EC2 Auto Scaling group. If you need to terminate instances, any tasks that run on them will also halt.

Scaling out a cluster does not also increase the running task count. You use service automatic scaling for this process.

AWS Fargate

AWS Fargate simplifies the process of managing containers in your environment and removes the need to manage underlying cluster instances. Instead, you only need to specify the compute requirements of your containers in your task definition. AWS Fargate automatically launches containers without your interaction.

With AWS Fargate, there are several restrictions on the types of tasks that you can launch. For example, when you specify a task definition, containers cannot be run in privileged mode. To verify that a given task definition is acceptable by AWS Fargate, use the **Requires** capabilities field of the Amazon ECS console or the `--requires-capabilities` command option of the AWS CLI.

AWS Fargate requires that containers launch with the network mode set to `awsvpc`. In other words, you can launch only AWS Fargate containers into Amazon VPCs.

AWS Fargate requires the `awslogs` driver to enable log configuration.

Containers and Images

Amazon ECS launches and manages Docker containers. However, Docker is not in scope for the AWS Certified Developer – Associate Exam.

Any workloads that run on Amazon ECS must reside in Docker containers. In a virtual server environment, multiple virtual machines share physical hardware, each of which acts as its own operating system. In a containerized environment, you package components of the operating system itself into containers. This removes the need to run any nonessential aspects of a full-fledged virtual machine to increase portability. In other words, virtual machines share the same physical hardware, while containers share the same operating system.

Container images are similar in concept to AMIs. Images provision a Docker container. You store images in registries, such as a Docker Hub or an Amazon Elastic Container Repository (ECR).

You can create your own private image repository; however, AWS Fargate does not support this launch type.

Docker provides mobility and flexibility of your workload to allow containers to be run on any system that supports Docker. Compute resources can be better utilized when you run multiple containers on the same cluster, which makes the best possible use of resources and reduces idle compute capacity. Since you separate service components into containers, you can update individual components more frequently and at reduced risk.

Task Definition

Though you can package entire applications into a single container, it may be more efficient to run multiple smaller containers, each of which contains a subset of functionality of your full application. This is referred to as *service-oriented architecture* (SOA). In SOA, each unit of functionality for an overall system is contained separately from the rest. Individual services work with one another to perform a larger task. For example, an e-commerce website that uses SOA could have sets of containers for load balancing, credit card processing, order fulfillment, or any other tasks that users require. You design each component of the system as a black box so that other components do not need to be aware of inner workings to interact with them.

A *task definition* is a JSON document that describes what containers launch for your application or system. A single task definition can describe between one and 10 containers and their requirements. Task definitions can also specify compute, networking, and storage

requirements, such as which ports to expose to which containers and which volumes to mount.

You should add containers to the same task definition under the following circumstances:

- The containers all share a common lifecycle.
- The containers need to run on the same common host or container instance.
- The containers need to share local resources or volumes.

An entire application does not need to deploy with a single task definition. Instead, you should separate larger application segments into separate task definitions. This will reduce the impact of breaking changes in your environment. If you allocate the right-sized container instances, you can also better control scaling and resource consumption of the containers.

After a task definition creates and uploads to Amazon ECS, it can launch one or more *tasks*. When a task is created, the containers in the task definition are scheduled to launch into the target cluster via the task scheduler.

Task Definition with Two Containers

The following example demonstrates a task definition with two containers. The first container runs a WordPress installation and binds the container instance's port 80 to the same port on the container. The second container installs MySQL to act as the backend data store of the WordPress container. The task definition also specifies a link between the containers, which allows them to communicate without port mappings if the network setting for the task definition is set to `bridge`.

```
{
  "containerDefinitions": [
    {
      "name": "wordpress",
      "links": [
        "mysql"
      ],
      "image": "wordpress",
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
      ],
```

(continued)

(continued)

```
      "memory": 500,
      "cpu": 10
    },
    {
      "environment": [
        {
          "name": "MYSQL_ROOT_PASSWORD",
          "value": "password"
        }
      ],
      "name": "mysql",
      "image": "mysql",
      "cpu": 10,
      "memory": 500,
      "essential": true
    }
  ],
  "family": "hello_world"
}
```

Services

When creating a *service*, you can specify the task definition and number of tasks to maintain at any point in time. After the service creates, it will launch the desired number of tasks; thus, it launches each of the containers in the task definition. If any containers in the task become unhealthy, the service is responsible and launches replacement tasks.

Deployment Strategies

When you define a service, you can also configure deployment strategies to ensure a minimum number of healthy tasks are available to serve requests while other tasks in the service update. The `maximumPercent` parameter defines the maximum percentage of tasks that can be in `RUNNING` or `PENDING` state. The `minimumHealthyPercent` parameter specifies the minimum percentage of tasks that must be in a healthy (`RUNNING`) state during deployments.

Suppose you configure one task for your service, and you would like to ensure that the application is available during deployments. If you set the `maximumPercent` to 200 percent and `minimumHealthyPercent` to 100 percent, it will ensure that the new task launches before the old task terminates. If you configure two tasks for your service and some loss of availability is acceptable, you can set `maximumPercent` to 100 percent and `minimumHealthyPercent` to 50 percent. This will cause the service scheduler to terminate one task, launch its replacement, and then do the same with the other task. The difference is that the first approach requires double the normal cluster capacity to accommodate the additional tasks.

Balance Loads

You can configure services to run behind a load balancer to distribute traffic automatically to tasks in the service. Amazon ECS supports classic load balancers, application load balancers, and network load balancers to distribute requests. Of the three load balancer types, application load balancers provide several unique features.

Application Load Balancing (ALB) load balancers route traffic at layer 7 (HTTP/HTTPS). Because of this, they can take advantage of dynamic host port mapping when you use them in front of Amazon ECS clusters. ALBs also support path-based routing so that multiple services can listen on the same port. This means that requests will be to different tasks based on the path specified in the request.

Classic load balancers, because they register and deregister instances, require that any tasks being run behind the load balancer all exist on the same container instance. This may not be desirable in some cases, and it would be better to use an ALB.

Schedule Tasks

If you increase the number of instances in an Amazon ECS cluster, it does not automatically increase the number of running tasks as well. When you configure a service, the service scheduler determines how many tasks run on one or more clusters and automatically starts replacement tasks should any fail. This is especially ideal for long-running tasks such as web servers. If you configure it to do so, the service scheduler will ensure that tasks register with an elastic load balancer.

You can also run a task manually with the `RunTask` action, or you can run tasks on a `cron`-like schedule (such as every *N* minutes on Tuesdays and Thursdays). This works well for tasks such as log rotation, batch jobs, or other data aggregation tasks.

To dynamically adjust the run task count dynamically, you use Amazon CloudWatch Alarms in conjunction with Application Auto Scaling to increase or decrease the task count based on alarm status. You can use two approaches for automatically scaling Amazon ECS services and tasks: Target Tracking Policies and Step Scaling Policies.

Target Tracking Policies

Target tracking policies determine when to scale the number of tasks based on a target metric. If the metric is above the target, such as CPU utilization being above 75 percent, Amazon ECS can automatically launch more tasks to bring the metric below the desired value. You can specify multiple target tracking policies for the same service. In the case of a conflict, the policy that would result in the highest task count wins.

Step Scaling Policies

Unlike target tracking policies, *step scaling policies* can continue to scale in or out as metrics increase or decrease. For example, you can configure a step scaling policy to scale out when CPU utilization reaches 75 percent, again at 80 percent, and one final time at 90 percent. With this approach, a single policy can result in multiple scaling activities as metrics increase or decrease.

Task Placement Strategies

Regardless of the method you use, *task placement strategies* determine on which instances tasks launch or which tasks terminate during scaling actions. For example, the `spread` task placement strategy distributes tasks across multiple AZs as much as possible. Task placement strategies perform on a best-effort basis. If the strategy cannot be honored, such as when there are insufficient compute resources in the AZ you select, Amazon ECS will still try to launch the task(s) on other cluster instances. Other strategies include `binpack` (uses CPU and memory on each instance at a time) and `random`.

Task placement strategies associate with specific `attributes`, which are evaluated during task placement. For example, to spread tasks across availability zones, the placement strategy to use is as follows:

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    }
]
```

Task Placement Constraints

Task placement constraints enforce specific requirements on the container instances on which tasks launch, such as to specify the instance type as `t2.micro`.

```
"placementConstraints": [
    {
        "expression": "attribute:ecs.instance-type == t2.micro",
        "type": "memberOf"
    }
]
```

Amazon ECS Service Discovery

Amazon ECS Service Discovery allows you to assign Amazon Route 53 DNS entries automatically for tasks your service manages. To do so, you create a private service namespace for each Amazon ECS cluster. As tasks launch or terminate, the private service namespace updates to include DNS entries for each task. A service directory maps DNS entries to available service endpoints. Amazon ECS Service Discovery maintains health checks of containers, and it removes them from the service directory should they become unavailable.

To use public namespaces, you must purchase or register the public hosted zone with Amazon Route 53.

Private Image Repositories

Amazon ECS can connect to private image repositories with basic authentication. This is useful to connect to Docker Hub or other private registries with a username and password. To do so, the `ECS_ENGINE_AUTH_TYPE` and `ECS_ENGINE_AUTH_DATA` environment variables must be set with the authorization type and actual credentials to connect. However, you should not set these properties directly. Instead, store your container instance configuration file in an Amazon S3 bucket and copy it to the instance with `userdata`.

Amazon Elastic Container Repository

Amazon Elastic Container Repository (Amazon ECR) is a Docker registry service that is fully compatible with existing Docker CLI tools. Amazon ECR supports resource-level permissions for private repositories and allows you to preserve a secure registry without the need to maintain an additional application. Since it integrates with IAM users and Amazon ECS cluster instances, it can take advantage of IAM users or instance profiles to access and maintain images securely without the need to provide a username and password.

Amazon ECS Container Agent

The *Amazon ECS container agent* is responsible for monitoring the status of tasks that run on cluster instances. If a new task needs to launch, the container agent will download the container images and start or stop containers. If any containers fail health checks, the container agent will replace them. Since the AWS Fargate launch type uses AWS-managed compute resources, you do not need to configure the agent.

To register an instance with an Amazon ECS cluster, you must first install the Amazon ECS Agent. This agent installs automatically on Amazon ECS optimized AMIs. If you would like to use a custom AMI, it must adhere to the following requirements:

- Linux kernel 3.10 or greater
- Docker version 1.9.0 or greater and any corresponding dependencies

The Amazon ECS container agent updates regularly and can update on your instance(s) without any service interruptions. To perform updates to the agent, replace the container instance entirely or use the Update Container Agent command on Amazon ECS optimized AMIs.

You cannot perform agent updates on Windows instances using these methods. Instead, terminate the instance and create a new server in its absence.

To configure the Amazon ECS container agent, update `/etc/ecs/config` on the container instance and then restart the agent. You can configure properties such as the cluster to register with, reserved ports, proxy settings, and how much system memory to reserve for the agent.

Amazon ECS Service Limits

Table 9.4 displays the limits that AWS enforces for Amazon ECS. You can change limits with an asterisk (*) by making a request to AWS Support.

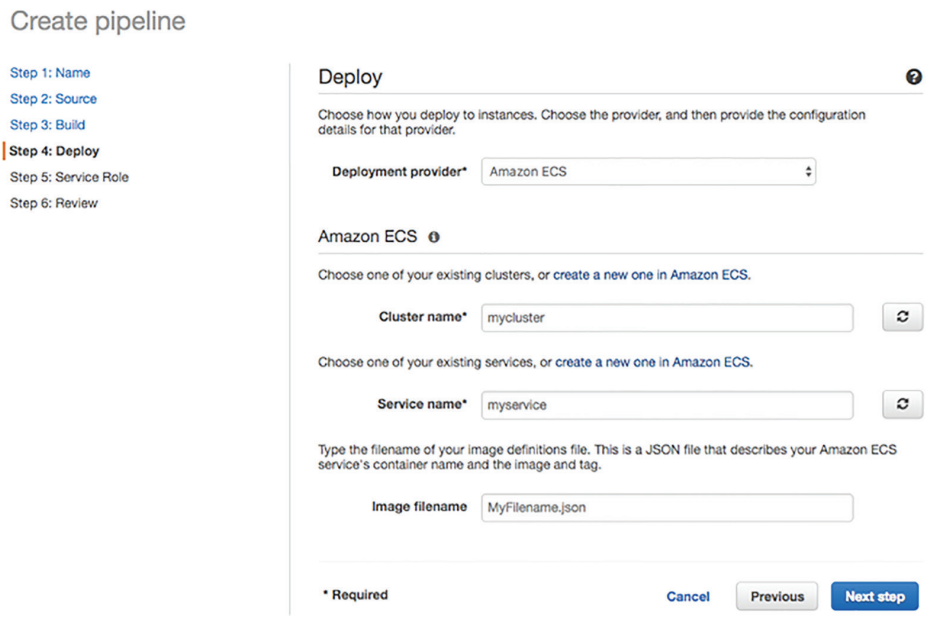
TABLE 9.4 Amazon ECS Service Limits

Limit	Value
Clusters per region per account*	1,000
Container instances per cluster*	1,000
Services per cluster*	500
Tasks that use Amazon EC2 launch type per service*	1,000
Tasks that use AWS Fargate launch type per region per account*	20
Public IP addresses for tasks that use AWS Fargate launch type*	20
Load balancers per service	1
Task definition size	32 KiB
Task definition containers	10
Layer size of image that use AWS Fargate task	4 GB
Shared volume that use AWS Fargate tasks	10 GB
Container storage that use AWS Fargate tasks	10 GB

Using Amazon ECS with AWS CodePipeline

When you select Amazon ECS as a deployment provider, there is no option to create the cluster and service as part of the pipeline creation process. This must be done ahead of time. After the cluster is created, select the appropriate cluster and service names in the AWS CodePipeline console, as shown in Figure 9.16.

FIGURE 9.16 Amazon ECS as a deployment provider



You must provide an image filename as part of this configuration. This is a JSON-formatted document inside your code repository or archive or as an output build artifact, which specifies the service's container name and image tag. We recommend that the cluster contain at least two Amazon EC2 instances so that one can act as primary while the other handles deployment of new containers.

Summary

This chapter includes infrastructure, configuration, and deployment services that you use to deploy configuration as code.

AWS CloudFormation leverages standard AWS APIs to provision and update infrastructure in your account. AWS CloudFormation uses standard configuration management tools such as Chef and Puppet.

Configuration management of infrastructure over an extended period of time is best served with the use of a dedicated tool such as AWS OpsWorks Stacks. You define the configuration in one or more Chef recipes to achieve configuration as code on top of your infrastructure. AWS OpsWorks Stacks can be used to provide a serverless Chef infrastructure to configure servers with Chef code (recipes).

Chef recipe code is declarative in nature, and you do not have to rely on the accuracy of procedural steps, as you would with a `userdata` script you apply to Amazon ECS instances or launch configurations. You can use Amazon ECS instead of instances or serverless functions to use a containerization method to manage applications. If you separate infrastructure from configuration, you also gain the ability to update each on separate cadences.

Amazon ECS supports Docker containers, and it allows you to run and scale containerized applications on AWS. Amazon ECS eliminates the need to install and operate your own container orchestration software, manage and scale a cluster of virtual machines, or schedule containers on those virtual machines.

AWS Fargate reduces management further as it deploys containers to serverless architecture and removes cluster management requirements. To create a cluster and deploy services, you configure the resource requirements of containers and availability requirements. Amazon ECS manages the rest through an agent that runs on cluster instances. AWS Fargate requires no agent management.

Amazon ECS clusters are the foundational infrastructure components on which containers run. Clusters consist of Amazon EC2 instances in your Amazon VPC. Each cluster instance has an agent installed that is responsible for receiving scheduling/shutdown commands from the Amazon ECS service and reporting the current health status of containers (restart or replace).

In lieu of custom JSON, Chef 12.0 stacks support data bags to provide better compatibility with community cookbooks. You can declare data bags in the custom JSON field of the stack, layer, and deployment configurations to provide instances in your stack for any additional data that you would like to provide.

AWS OpsWorks Stacks lets you manage applications and servers on AWS and on-premises. You can model your application as a stack that contains different layers, such as load balancing, database, and application server. You can deploy and configure Amazon EC2 instances in each layer or connect other resources such as Amazon RDS databases. AWS OpsWorks Stacks lets you set automatic scaling for your servers on preset schedules or in response to a constant change of traffic levels, and it uses lifecycle hooks to orchestrate changes as your environment scales. You run Chef recipes with Chef Solo, which allows you to automate tasks such as installing packages and program languages or frameworks, configuring software, and more.

An app is the location where you store application code and other files, such as an Amazon S3 bucket, a Git repository, or an HTTP bundle, and it includes sign-in credentials. The `Deploy` lifecycle event includes any apps that you configure for an instance at the layer or layers to which it corresponds.

At each layer of a stack, you set which Chef recipes to execute at each stage of a node's lifecycle, such as when it comes online or goes offline (lifecycle events). The recipes at each lifecycle event are executed by the AWS OpsWorks Agent in the order you specify.

AWS OpsWorks Stacks allows for management of other resources in your account as part of your stack and include elastic IP addresses, Amazon EBS volumes, and Amazon RDS instances.

The AWS OpsWorks Stacks dashboard monitors up to 13 custom metrics for each instance in the stack. The agent that runs on each instance will publish the information to the AWS OpsWorks Stacks service. If you enable the layer, system, application, and custom logs, they automatically publish to Amazon CloudWatch Logs for review without accessing the instance itself.

When you define a consistent deployment pattern for infrastructure, configuration, and application code, you can convert entire enterprises to code. You can remove manual management of most common processes and replace them with seamless management of entire application stacks through a simple commit action.

Exam Essentials

Understand configuration management and Chef. Configuration management is the process designed to ensure the infrastructure in a given system adheres to a specific set of standards, settings, or attributes. Chef is a Ruby-based configuration management language that AWS OpsWorks Stacks uses to enforce configuration on Amazon EC2 on-premises instances, or *nodes*. Chef uses a declarative syntax to describe the desired state of a node, abstracting the actual steps needed to achieve the desired configuration. This code is organized into *recipes*, which are organized into collections called *cookbooks*.

Know how AWS OpsWorks Stacks organizes configuration code into cookbooks. In traditional Chef implementations, cookbooks belong to a `chef-repo`, which is a versioned directory that contains cookbooks and their underlying recipes and files. A single cookbook repository can contain one or more cookbooks. When you define the custom cookbook location for a stack, all cookbooks copy to instances in the stack.

Know how to update custom cookbooks on a node. When instances first launch in a stack, they will download cookbooks from the custom cookbook repository. You must manually issue an `Update Custom Cookbooks` command to instances in your stack to update the instance.

Understand the different AWS OpsWorks Stacks components. The topmost object in AWS OpsWorks Stacks is a stack, which contains all elements of a given environment or system. Within a stack, one or more layers contain instances you group by common purpose. A single instance references either an Amazon EC2 or on-premises instance and contains additional configuration data. A stack can contain one or more apps, which refer to repositories where application code copies to for deployment. Users are regional resources that you can configure to access one or more stacks in an account.

Know the different AWS OpsWorks Stacks instance types and their purpose. AWS OpsWorks Stacks has three different instance types: 24/7, time-based, and load-based. The 24/7 instances run continuously unless an authorized user manually stops it, and they are useful for handling the minimum expected load of a system. Time-based instances start and stop on a given 24-hour schedule and are recommended for predicable increases in load at

different times of the day. Load-based instances start and stop in response to metrics, such as CPU utilization for a layer, and you use them to respond to sudden increases in traffic.

Understand how AWS OpsWorks Stacks implements auto healing. The AWS OpsWorks Stacks agent that runs on an instance performs a health check every minute and sends the response to AWS. If the AWS OpsWorks Stacks agent does not receive the health check for five continuous minutes, the instance restarts automatically. You can disable this feature. Auto healing events publish to Amazon CloudWatch for reference.

Understand the AWS OpsWorks Stacks permissions model. AWS OpsWorks Stacks provides the ability to manage users at the stack level, independent of IAM permissions. This is useful for providing access to instances in a stack but not to the AWS Management Console or API. You can assign AWS OpsWorks Stacks users to one of four permission levels: `Deny`, `Show`, `Deploy`, and `Manage`. Additionally, you can give users SSH/RDP access to instances in a stack (with or without sudo/administrator permission). AWS OpsWorks Stacks users are regional resources. If you would like to give a user in one region access to a stack in another region, you need to copy the user to the second region. Some AWS OpsWorks Stacks activities are available only through IAM permissions, such as to delete and create stacks.

Know the different AWS OpsWorks Stacks lifecycle events. Instances in a stack are provisioned, configured, and retired using lifecycle events. The AWS OpsWorks Stacks supports the lifecycle events: `Setup`, `Configure`, `Deploy`, `Undeploy`, and `Shutdown`. The `Configure` event runs on all instances in a stack any time one instance comes online or goes offline.

Know the components of an Amazon ECS cluster. A cluster is the foundational infrastructure component on which containers are run. Clusters are made up of one or more Amazon EC2 instances, or they can be run on AWS-managed infrastructure using AWS Fargate. A task definition is a JSON file that describes which containers to launch on a cluster. Task definitions can be defined by grouping containers that are used for a common purpose, such as for compute, networking, and storage requirements. A service launches on a cluster and specifies the task definition and number of tasks to maintain. If any containers become unhealthy, the service is responsible for launching replacements.

Know the difference between Amazon ECS and AWS Fargate launch types. The AWS Fargate launch type uses AWS-managed infrastructure to launch tasks. As a customer, you are no longer required to provision and manage cluster instances. With AWS Fargate, each cluster instance is assigned a network interface in your VPC. Amazon ECS launch types require a cluster in your account, which you must manage over time.

Know how to scale running tasks in a cluster. Changing the number of instances in a cluster does not automatically cause the number of running tasks to scale in or out. You can use target tracking policies and step scaling policies to scale tasks automatically based on target metrics. A target tracking policy determines when to scale based on metrics such as CPU utilization or network traffic. Target tracking policies keep metrics within a certain boundary. For example, you can launch additional tasks if CPU utilization is above 75 percent. Step scaling policies can continuously scale as metrics increase or decrease. You can configure a step scaling policy to scale tasks out when CPU utilization reaches 75 percent

and again at 80 percent and 90 percent. A single step scaling policy can result in multiple scaling activities.

Know how images are stored in Amazon Elastic Container Repository (Amazon ECR). Amazon ECR is a Docker registry service that is fully compatible with existing Docker tools. Amazon ECR supports resource-level permissions for private repositories, and it allows you to maintain a secure registry without the need to maintain additional instances/applications.

Resources to Review

Continuous Deployment to Amazon ECS with AWS CodePipeline, AWS CodeBuild, Amazon ECR, and AWS CloudFormation:

`https://aws.amazon.com/blogs/compute/continuous-deployment-to-amazon-ecs-using-aws-codepipeline-aws-codebuild-amazon-ecr-and-aws-cloudformation/`

How to set up AWS OpsWorks Stacks auto healing notifications in Amazon CloudWatch Events:

`https://aws.amazon.com/blogs/mt/how-to-set-up-aws-opsworks-stacks-auto-healing-notifications-in-amazon-cloudwatch-events/`

Managing Multi-Tiered Applications with AWS OpsWorks:

`https://d0.awsstatic.com/whitepapers/managing-multi-tiered-web-applications-with-opsworks.pdf`

AWS OpsWorks Stacks:

`https://aws.amazon.com/opsworks/stacks/`

How do I implement a configuration management solution on AWS?:

`https://aws.amazon.com/answers/configuration-management/aws-infrastructure-configuration-management/`

Docker on AWS:

`https://d1.awsstatic.com/whitepapers/docker-on-aws.pdf`

What are Containers?

`https://aws.amazon.com/containers/`

Amazon Elastic Container Service (ECS):

`https://aws.amazon.com/ecs/`

You can access the features of AWS Auto Scaling using the AWS CLI, which provides commands to use with Amazon EC2 and Amazon CloudWatch and Elastic Load Balancing.

To scale a resource other than Amazon EC2, you can use the Application Auto Scaling API, which allows you to define scaling policies to scale your AWS resources automatically or schedule one-time or recurring scaling actions.

Using Containers

Containers provide a standard way to package your application's code, configurations, and dependencies into a single object. Containers share an operating system installed on the server and run as resource-isolated processes, ensuring quick, reliable, and consistent deployments, regardless of environment.

Containers provide process isolation that lets you granularly set CPU and memory utilization for better use of compute resources.

Containerize Everything

Containers are a powerful way for developers to package and deploy their applications. They are lightweight and provide a consistent, portable software environment for applications to run and scale effortlessly anywhere.

Use Amazon Elastic Container Service (Amazon ECS) to build all types of containerized applications easily, from long-running applications and microservices to batch jobs and machine learning applications. You can migrate legacy Linux or Windows applications from on-premises to the AWS Cloud and run them as containerized applications using Amazon ECS.

Amazon ECS enables you to use containers as building blocks for your applications by eliminating the need for you to install, operate, and scale your own cluster management infrastructure. You can schedule long-running applications, services, and batch processes using Docker containers. Amazon ECS maintains application availability and allows you to scale your containers up or down to meet your application's capacity requirements. Amazon ECS is integrated with familiar features like Elastic Load Balancing, EBS volumes, virtual private cloud (VPC), and AWS Identity and Access Management (IAM). Use APIs to integrate and use your own schedulers or connect Amazon ECS into your existing software delivery process.

Containers without Servers

AWS Fargate technology is available with Amazon ECS. With Fargate, you no longer have to select Amazon EC2 instance types, provision and scale clusters, or patch and update each server. You do not have to worry about task placement strategies, such as binpacking or host spread, and tasks are automatically balanced across Availability Zones. Fargate manages the availability of containers for you. You define your application's requirements,

select Fargate as your launch type in the AWS Management Console or AWS CLI, and Fargate takes care of all of the scaling and infrastructure management required to run your containers.

For developers who require more granular, server-level control over the infrastructure, Amazon ECS EC2 launch type enables you to manage a cluster of servers and schedule placement of containers on the servers.

Using Serverless Approaches

Serverless approaches are ideal for applications whereby load can vary dynamically. Using a serverless approach means no compute costs are incurred when there is no user traffic, while still offering instant scale to meet high demand, such as a flash sale on an ecommerce site or a social media mention that drives a sudden wave of traffic. All of the actual hardware and server software are handled by AWS.

Benefits gained by using AWS Serverless services include the following:

- No need to manage servers
- No need to ensure application fault tolerance, availability, and explicit fleet management to scale to peak load
- No charge for idle capacity

You can focus on product innovation and rapidly construct these applications:

- Amazon S3 offers a simple hosting solution for static content.
- AWS Lambda, with Amazon API Gateway, supports dynamic API requests using functions.
- Amazon DynamoDB offers a simple storage solution for session and per-user state.
- Amazon Cognito provides a way to handle user registration, authentication, and access control to resources.
- AWS Serverless Application Model (AWS SAM) can be used by developers to describe the various elements of an application.
- AWS CodeStar can set up a CI/CD toolchain with a few clicks.

Compared to traditional infrastructure approaches, an application is also often less expensive to develop, deliver, and operate when it has been architected in a serverless fashion. The serverless application model is generic, and it applies to almost any type of application from a startup to an enterprise.

Here are a few examples of application use cases:

- Web applications and websites
- Mobile backends
- Media and log processing

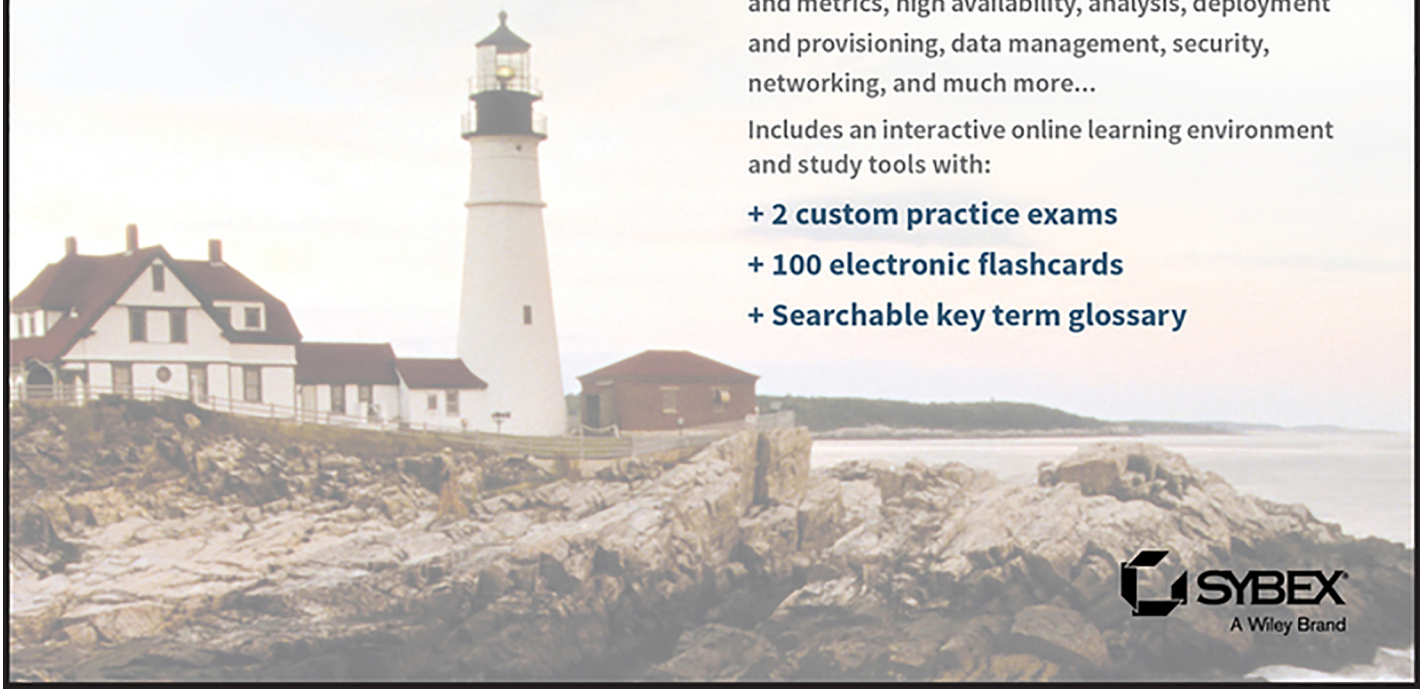
amazon web services

Stephen Cole, Gareth Digby, Chris Fitch,
Steve Friedberg, Shaun Qualheim, Jerry Rhoads,
Michael Roth, Blaine Sundrud

AWS Certified SysOps Administrator

OFFICIAL STUDY GUIDE

ASSOCIATE EXAM



Covers exam objectives, including monitoring and metrics, high availability, analysis, deployment and provisioning, data management, security, networking, and much more...

Includes an interactive online learning environment and study tools with:

+ 2 custom practice exams

+ 100 electronic flashcards

+ Searchable key term glossary

SYBEX
A Wiley Brand

groups to control access on a specific interface. With administrative access, you again can control access with placement of Amazon EC2 instances in an Amazon VPC, the use of security groups, and the use of public/private key pairs both to encrypt traffic and control access.

Controlling Administrative Access

AWS provides a number of tools to manage the security of your instances. These tools include the following:

- IAM
- AWS Trusted Advisor
- AWS Service Catalog
- Amazon Inspector

IAM allows you to create policies that control access to APIs and apply those policies to users, groups, and roles.

AWS Trusted Advisor, which comes as part of AWS Support, looks at things like open ports on security groups and level of access allowed to Amazon EC2 instances, and it makes recommendations to improve the security of your AWS infrastructure.

AWS Service Catalog allows IT administrators to create, manage, and distribute portfolios of approved products to end users who can then access the products they need in a personalized portal. AWS Service Catalog allows organizations to manage commonly deployed IT services centrally, and it helps organizations achieve consistent governance and meet compliance requirements while enabling users to deploy quickly only the approved IT services they need within the constraints the organization sets.

Amazon Inspector is a security vulnerability assessment service that helps improve the security and compliance of your AWS resources. Amazon Inspector automatically assesses resources for vulnerabilities or deviations from best practices and then produces a detailed list of security findings prioritized by level of severity.

AWS also has a number of developer documents, whitepapers, articles, and tutorials to help you in securing your environment. These are available on the Cloud Security Resources page of the AWS website.

Amazon EC2 Container Service (Amazon ECS)

Amazon ECS is a highly scalable container management service. Amazon ECS makes it easy to run, stop, and manage Docker containers on Amazon EC2 instances.

With Amazon ECS, you can launch and place containers across your cluster using API calls. You schedule the placement of containers based on your resource needs, isolation

policies, and availability requirements. You can use Amazon ECS both for cluster management and for scheduling deployments of containers onto hosts.

Amazon ECS eliminates the need for you to operate your own cluster management and configuration management systems. Amazon ECS can be used to manage and scale both batch and Extract, Transform, Load (ETL) workloads. It can also be used to build application architectures based on the microservices model.

Amazon ECS is a regionally-based service that can be used to run application containers in a highly available manner across all Availability Zones within an AWS Region.

Implementation

To implement Amazon ECS, you need to install an Amazon ECS agent on your Amazon EC2 instances. If you use Amazon ECS-optimized AMIs, that agent is already installed. Additionally, the container instance needs to have an IAM role that authenticates to your account and will need external network access to communicate with the Amazon ECS service endpoint.

Clusters are the logical grouping of container instances that you can place tasks on. Clusters are region specific and can contain multiple, different instance types (though an instance can only be a member of a single cluster).

Amazon ECS obtains a Docker image for repository. This repository can be on AWS or on other infrastructure.

Deploying a Container

To deploy a container, you need to do the following:

1. **Define a task**. This is where you assign the name, provide the image name (important for locating the correct image), and decide on the amount of memory and CPU needed.
2. **Define the service**. In this step, you decide how many instances of the task you want to run in the cluster and any Elastic Load Balancing load balancers that you want to register the instances with.
3. **Create the Amazon ECS cluster**. This is where the cluster is created and also where you specify the number of instances to run. The cluster can run across multiple Availability Zones.
4. **Create the stack**. A stack of instances is created based on the configuration information provided. You can monitor the creation of the stack in the AWS Management Console. Creation of the stack is usually completed in less than five minutes.

Management

Amazon ECS can be configured using the AWS Management Console, the AWS CLI, or the Amazon ECS CLI.

Monitoring Amazon ECS

The primary tool used for monitoring your Amazon ECS clusters is Amazon CloudWatch. Amazon CloudWatch collects Amazon ECS metric data in one-minute periods and sends them to Amazon CloudWatch. Amazon CloudWatch stores these metrics for a period of two weeks. You can monitor the CPU and memory reservation and utilization across your cluster as a whole and the CPU and memory utilization on the services in your cluster. You can use Amazon CloudWatch to trigger alarms, set notifications, and create dashboards to monitor the services.

Once it's set up, you can view Amazon CloudWatch metrics in both the Amazon ECS console and the Amazon CloudWatch console. The Amazon ECS console provides a maximum 24-hour view while the Amazon CloudWatch console provides a fine-grained and customizable view of running services.

The other tool available is AWS CloudTrail, which will log all Amazon ECS API calls.

AWS Trusted Advisor is another source for monitoring all of your AWS resources, including Amazon ECS, to improve performance, reliability, and security.

There is no additional cost for using Amazon ECS. The only charges are for the Amazon EC2 instances or AWS Lambda requests and compute time.

Security

With Amazon ECS, you need to do the following:

1. Control who can create task definitions.
2. Control who can deploy clusters.
3. Control who can access the Amazon EC2 instances.

IAM is the tool used for the first two necessities. For controlling access to Amazon EC2 instances, the tools described in the Amazon EC2 section still apply. You can use IAM roles, security groups, and (because these Amazon EC2 instances are located in an Amazon VPC) network Access Control Lists (ACLs) and route tables to control access to the Amazon EC2 instances.

AWS Elastic Beanstalk

AWS Elastic Beanstalk enables you to deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications. AWS Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and AWS Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

AWS Elastic Beanstalk provisions resources to support a web application that handles HTTP(S) requests or an application that handles background-processing tasks. An environment tier whose web application processes web requests is known as a *web server tier*. An environment tier whose application runs background jobs is known as a *worker tier*. The environment is the heart of the application. When you create an environment, AWS Elastic Beanstalk provisions the resources required to run your application.

If your application performs operations or workflows that take a long time to complete, you can offload those tasks to a dedicated worker environment. Decoupling your web application front end from a process that performs blocking operations is a common way to ensure that your application stays responsive under load. For the worker environment tier, AWS Elastic Beanstalk also creates and provisions an Amazon Simple Queue Service (Amazon SQS) queue if you don't already have one. When you launch a worker environment tier, AWS Elastic Beanstalk installs the necessary support files for your programming language of choice and a daemon on each Amazon EC2 instance in the Auto Scaling group. The daemon is responsible for pulling requests from an Amazon SQS queue and sending the data to the application running in the worker environment tier that will process those messages.

Environment Platforms

AWS Elastic Beanstalk supports Java, .NET, PHP, Node.js, Python, Packer, Ruby, Go, and Docker, and it is ideal for web applications. Due to AWS Elastic Beanstalk's open architecture, non-web applications can also be deployed using AWS Elastic Beanstalk. Additionally, AWS Elastic Beanstalk supports custom platforms that can be based on an AMI that you create from one of the supported operating systems and that can include further customizations. You can choose to have your AWS Elastic Beanstalk environments automatically updated to the latest version of the underlying platform running your application during a specified maintenance window. AWS Elastic Beanstalk regularly releases new versions of supported platforms with operating system, web and application server, language, and framework updates.

Managing Environments

AWS Elastic Beanstalk makes it easy to create new environments for your application. You can create and manage separate environments for development, testing, and production use, and you can deploy any version of your application to any environment. Environments can be long-running or temporary. When you terminate an environment, you can save its configuration to re-create it later.

As you develop your application, you will deploy it often, possibly to several different environments for different purposes. AWS Elastic Beanstalk lets you configure how deployments are performed. You can deploy to all of the instances in your environment simultaneously or split a deployment into batches with rolling deployments.

Managing Application Versions

AWS Elastic Beanstalk creates an application version whenever you upload source code. This usually occurs when you create a new environment or upload and deploy code using

the environment management console or AWS Elastic Beanstalk CLI. You can also upload a source bundle without deploying it from the application console.

A single-environment deployment of AWS Elastic Beanstalk performs an in-place update when you update your application versions, so your application may become unavailable to users for a short period of time. It is possible to avoid this downtime by performing a blue/green deployment with AWS Elastic Beanstalk, where you deploy the new version to a separate environment and then use AWS Elastic Beanstalk to swap the CNAMEs of the two environments to redirect traffic to the new version instantly.

Blue/green deployments require that your application's environments run independently of your production database—if your application uses one. If your AWS Elastic Beanstalk green environment has an Amazon RDS DB instance included in it, the data will not transfer over to the AWS Elastic Beanstalk blue environment and will be lost if you terminate the original environment.

Amazon EC2 Container Service

Amazon EC2 Container Service (Amazon ECS) is a highly scalable, high-performance container management service that makes it easy to run, stop, and manage Docker containers on a cluster of Amazon EC2 instances. *Docker* is a technology that allows you to build, run, test, and deploy distributed applications that are based on Linux containers. Amazon ECS uses Docker images in task definitions to launch containers on Amazon EC2 instances in your clusters.

Amazon ECS lets you launch and stop container-based applications with simple API calls, allowing you to get the state of your cluster from a centralized service and giving you access to many familiar Amazon EC2 features, such as security groups, Amazon EBS volumes, and IAM roles. Amazon ECS is a good option if you are using Docker for a consistent build and deployment experience or as the basis for sophisticated distributed systems. Amazon ECS is also a good option if you want to improve the utilization of your Amazon EC2 instances.

You can use Amazon ECS to schedule the placement of containers across your cluster based on your resource needs, isolation policies, and availability requirements. Amazon ECS eliminates the need for you to operate your own cluster management and configuration management systems or worry about scaling your management infrastructure. It also can be used to create a consistent deployment and build experience, manage and scale batch and Extract-Transform-Load (ETL) workloads, and build sophisticated application architectures on a microservices model.

While AWS Elastic Beanstalk can also be used to develop, test, and deploy Docker containers rapidly in conjunction with other components of your application infrastructure, using Amazon ECS directly provides more fine-grained control and access to a wider set of use cases.

Clusters

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 Amazon VPC. After a cluster is up and running, you can indicate 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. A *cluster* is a logical grouping of Amazon EC2 instances on which you run tasks. When running a task, Amazon ECS downloads your container images from a registry that you specify and runs those images on the container instances within your cluster.

Instances

An *Amazon ECS container instance* is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into a cluster. When you run tasks with Amazon ECS, your tasks are placed on your active container instances. A container instance must be running the Amazon ECS container agent to register into one of your clusters. If you are using the Amazon ECS-optimized AMI, the agent is already installed. To use a different operating system, install the appropriate agent.

The Amazon ECS container agent makes calls to Amazon ECS on your behalf. You must launch container instances with an instance profile that has an appropriate IAM role. The IAM role must have a policy attached that authenticates to your account and provides the required resource permissions. Additionally, the container instances need external network access to allow the agent to communicate with the Amazon ECS service endpoint. If your container instances do not have public IP addresses, then they must use Network Address Translation (NAT) or an HTTP proxy to provide this access. When the Amazon ECS container agent registers an instance into your cluster, the container instance reports its status as ACTIVE and its agent connection status as TRUE. This container instance can accept run task requests.

The Amazon ECS-optimized AMI is the recommended AMI for you to use to launch your Amazon ECS container instances. The Amazon ECS-optimized AMI is preconfigured and has been tested on Amazon ECS by AWS engineers. It provides the latest, tested, minimal version of the Amazon Linux AMI, Amazon ECS container agent, recommended version of Docker, and container services package to run and monitor the Amazon ECS agent. Typically, the Amazon ECS config file on the instance is modified with a user data script to specify parameters such as the cluster with which to register the instance.

Containers

To deploy applications on Amazon ECS, your application components must be architected to run in containers. A Docker container is a standardized unit of software development that contains everything that your software application needs to run including operating system, configuration, code, runtime, system tools, and system libraries. Containers are lighter in weight and have less memory and computational overhead than virtual machines, so they make it easy to support applications that consist of hundreds or thousands of small, isolated microservices. A properly containerized application is easy to scale and maintain and makes efficient use of available system resources.

Building your application as a collection of tight, focused containers allows you to build them in parallel with strict, well-defined interfaces. With better interfaces between microservices, you have the freedom to improve and even totally revise implementations without fear of breaking running code. Because your application's dependencies are spelled out explicitly and declaratively, less time will be lost diagnosing, identifying, and fixing issues that arise from missing or obsolete packages. Using containers allows you to build components that run in isolated environments, thus limiting the ability of one container to disrupt the operation of another accidentally while still being able to share libraries and other common resources cooperatively. This opportunistic sharing reduces memory pressure and leads to increased runtime efficiency.

Images

Containers are created from a read-only template called an *image*. Images are typically built from a *Dockerfile*, a manifest that describes the base image to use for your Docker image and what you want installed and running on it. When you build the Docker image from your Dockerfile, the images are stored in a registry from which they can be downloaded and run on your container instances.

Repository

Docker uses images that are stored in repositories to launch containers. The default repository for the Docker daemon is Docker Hub. Although you don't need a Docker Hub account to use Amazon ECS or Docker, having a Docker Hub account gives you the freedom to store your modified Docker images so that you can use them in your Amazon ECS task definitions.

Another registry option is Amazon EC2 Container Registry (Amazon ECR). Amazon ECR is a managed AWS Docker registry service. Customers can use the familiar Docker CLI to push, pull, and manage images. Amazon ECR supports private Docker repositories with resource-based permissions using IAM so that specific users or Amazon EC2 instances can access repositories and images.

Tasks

A *task definition* is like a blueprint for your application. Every time you launch a task in Amazon ECS, you specify a task definition so that the service knows which Docker image to use for containers, how many containers to use in the task, and the resource allocation for each container.

Amazon ECS provides three task features:

- A service scheduler for long-running tasks and applications
- The ability to run tasks manually for batch jobs or single-run tasks, with Amazon ECS placing tasks on your cluster for you
- The ability to run tasks on the container instance that you specify so that you can integrate with custom or third-party schedulers or place a task manually on a specific container instance

A task is the instantiation of a task definition on a container instance within your cluster. Before you can run Docker containers on Amazon ECS, you must create a task definition. You can define multiple containers and data volumes in a task definition.

Services

Amazon ECS allows you to run and maintain a specified number (the "desired count") of instances of a task definition simultaneously in an Amazon ECS cluster. This is called a *service*. If any of your tasks should fail or stop for any reason, the Amazon ECS service scheduler launches another instance of your task definition to replace it and maintain the desired count of tasks in the service. In addition to maintaining the desired count of tasks in your service, you can optionally run your service behind a load balancer. The load balancer distributes traffic across the tasks that are associated with the service.

The service scheduler is ideally suited for long-running, stateless services and applications. You can update your services that are maintained by the service scheduler, such as deploying a new task definition or changing the running number of desired tasks. By default, the service scheduler spreads tasks across Availability Zones, but you can use task placement strategies and constraints to customize task placement decisions.

AWS OpsWorks Stacks

AWS OpsWorks is an application management service that makes it easy for developers and operations personnel to deploy and operate applications of all shapes and sizes. AWS OpsWorks works best if you want to deploy your code, have some abstraction from the underlying infrastructure, and have an application more complex than a Three-Tier architecture. AWS OpsWorks is also recommended if you want to manage your infrastructure with a configuration management system such as Chef.

AWS OpsWorks Stacks provides a simple and flexible way to create and manage stacks and applications. It has a rich set of customizable components that you can mix and match to create a stack that satisfies your specific purposes. Additionally, if you have existing computing resources, you can incorporate them into a stack along with instances that you created with AWS OpsWorks Stacks. Such resources can be existing Amazon EC2 instances or even on-premises instances that are running on your own hardware. You can then use AWS OpsWorks Stacks to manage all related instances as a group, regardless of how they were created.

Stacks

The *stack* is the top-level AWS OpsWorks Stacks entity. It represents a set of instances that you want to manage collectively, typically because they have a common purpose such as serving PHP applications. In addition to serving as a container, a stack handles tasks that apply to the group of instances as a whole, such as managing applications and cookbooks. For example, a stack whose purpose is to serve web applications might contain a set of application server instances, a load balancer to direct traffic to the instances, and a database instance that serves as a back end of the application instances. A common practice is to have multiple stacks that represent different environments (such as development, staging, and production stacks).

JON BONSO AND KENNETH SAMONTE

AWS CERTIFIED DEVOPS ENGINEER PROFESSIONAL

Tutorials Dojo
Study Guide and Cheat Sheets

CloudFormation Template for ECS, Auto Scaling and ALB

Amazon Elastic Container Service (ECS) allows you to manage and run Docker containers on clusters of EC2 instances. You can also configure your ECS to use Fargate launch type which eliminates the need to manage EC2 instances.

With CloudFormation, you can define your ECS clusters and tasks definitions to easily deploy your containers. For high availability of your Docker containers, ECS clusters are usually configured with an auto scaling group behind an application load balancer. These resources can also be declared on your CloudFormation template.

DevOps Exam Notes:

Going on to the exam, be sure to remember the syntax needed to declare your ECS cluster, Auto Scaling group, and application load balancer. The **AWS::ECS::Service** resource creates an ECS cluster and the **AWS::ECS::TaskDefinition** resource creates a task definition for your container. The **AWS::ElasticLoadBalancingV2::LoadBalance**r resource creates an application load balancer and the **AWS::AutoScaling::AutoScalingGroup** resource creates an EC2 auto scaling group.

AWS provides an example template which you can use to deploy a web application in an Amazon ECS container with auto scaling and application load balancer. Here's a snippet of the template with the core resources:

```
{
  "AWSTemplateFormatVersion":"2010-09-09",
  "Resources":{
      "ECSCluster":{
      "Type":"AWS::ECS::Cluster"

      },

……

      "taskdefinition":{
      "Type":"AWS::ECS::TaskDefinition",

      "Properties":{

……

      "ECSALB":{
      "Type":"AWS::ElasticLoadBalancingV2::LoadBalancer",
```

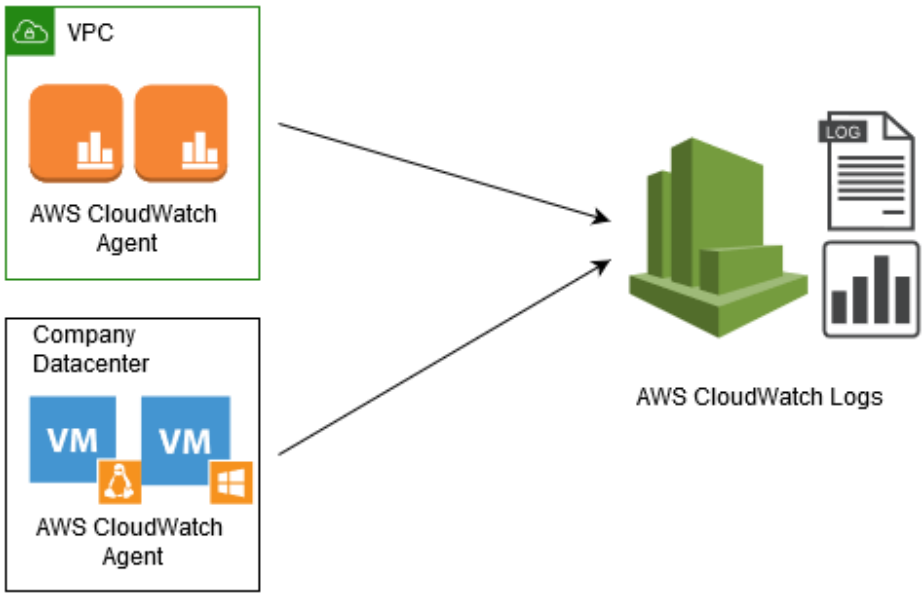
```
        "Properties":{

……

        "ECSAutoScalingGroup":{
        "Type":"AWS::AutoScaling::AutoScalingGroup",
        "Properties":{
        "VPCZoneIdentifier":{
              "Ref":"SubnetId"

        },
```

Sources:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/quickref-ecs.html
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ecs-service.html
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ecs-service-loadbalancers.html

Fetching Application Logs from Amazon EC2, ECS and On-premises Servers

Application logs are vital for monitoring, troubleshooting, and regulatory compliance of every enterprise system. Without it, your team will waste a lot of time trying to find the root cause of an issue that can be easily detected by simply checking the logs. These files often live inside the server or a container. Usually, you have to connect to the application server via SSH or RDP before you can view the logs. This manual process seems to be inefficient, especially for high-performance organizations with hybrid network architectures.

Using the Amazon CloudWatch Logs agent, you can collect system metrics and logs from your Amazon EC2 instances and on-premises application servers. Gone are the days of spending several minutes connecting to your server and manually retrieving the application logs. For Linux servers, you don't need to issue a *tail -f* command anymore since you can view the logs on the CloudWatch dashboard on your browser in near real-time. It also collects both system-level and custom metrics from your EC2 instances and on-premises servers, making your monitoring tasks a breeze.



You have to manually download and install the Amazon CloudWatch Logs agent to your EC2 instances or on-premises servers using the command line. Alternatively, you can use AWS Systems Manager to automate the installation process. For your EC2 instances, it is preferable to attach an IAM Role to allow the application to send data to CloudWatch. For your on-premises servers, you have to create a separate IAM User to integrate your server to CloudWatch. Of course, you should first establish a connection between your on-premises data center and VPC using a VPN or a Direct Connect connection. You have to use a named profile in your local server that contains the credentials of the IAM user that you created.

If you are running your containerized application in Amazon Elastic Container Service (ECS), you can view the different logs from your containers in one convenient location by integrating Amazon CloudWatch. You can configure your Docker containers' tasks to send log data to CloudWatch Logs by using the *awslogs* log driver.


```
"logConfiguration": {
    "logDriver": "awslogs",
    "options": {
        "awslogs-group": "awslogs-wordpress",
        "awslogs-region": "us-west-2",
        "awslogs-stream-prefix": "awslogs-example"
    }
```

If your ECS task is using a Fargate launch type, you can enable the *awslogs* log driver and add the required *logConfiguration* parameters to your task definition. For EC2 launch types, you have to ensure that your Amazon ECS container instances have an attached IAM role that contains *logs:CreateLogStream* and *logs:PutLogEvents* permissions. Storing the log files to Amazon CloudWatch prevents your application logs from taking up disk space on your container instances.

Sources:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/UseCloudWatchUnifiedAgent.html
https://docs.aws.amazon.com/AmazonECS/latest/userguide/using_awslogs.html

Amazon Elastic Container Service (ECS)

- A container management service to run, stop, and manage Docker containers on a cluster.
- ECS can be used to create a consistent deployment and build experience, manage, and scale batch and **Extract-Transform-Load** (ETL) workloads, and build sophisticated application architectures on a microservices model.

Features

- You can create 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.

Components

- Containers and Images
  - Your application components must be architected to run in **containers** — containing everything that your software application needs to run: code, runtime, system tools, system libraries, etc.
  - Containers are created from a read-only template called an **image**.
- Task Components
  - **Task definitions** specify various parameters for your application. It is a text file, in JSON format, that describes one or more containers, up to a maximum of ten, that form your application.
  - Task definitions are split into separate parts:
    - Task family - the name of the task, and each family can have multiple revisions.
    - IAM task role - specifies the permissions that containers in the task should have.
    - Network mode - determines how the networking is configured for your containers.
    - Container definitions - specify which image to use, how much CPU and memory the container are allocated, and many more options.
- 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, you can specify the number of tasks that will run on your cluster.
    - Each task that uses the Fargate launch type has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.
  - You can upload a new version of your application task definition, and the ECS scheduler automatically starts new containers using the updated image and stop containers running the previous version.
- Clusters
  - When you run tasks using ECS, you place them in a **cluster**, which is a logical grouping of resources.
  - Clusters can contain tasks using both the Fargate and EC2 launch types.

  - When using the Fargate launch type with tasks within your cluster, ECS manages your cluster resources.
  - Enabling managed Amazon ECS cluster auto scaling allows ECS to manage the scale-in and scale-out actions of the Auto Scaling group.
- Services
  - ECS allows you to run and maintain a specified number of instances of a task definition simultaneously in a cluster.
  - In addition to maintaining the desired count of tasks in your service, you can optionally run your service behind a load balancer.
  - There are two deployment strategies in ECS:
    - **Rolling Update**
      - This involves the service scheduler replacing the current running version of the container with the latest version.

    Blue/Green Deployment with AWS CodeDeploy
      - This deployment type allows you to verify a new deployment of a service before sending production traffic to it.
      - The service must be configured to use either an Application Load Balancer or Network Load Balancer.
- Container Agent (AWS ECS Agent)
  - The **container agent** runs on each infrastructure resource within an ECS cluster.
  - It sends information about the resource's current running tasks and resource utilization to ECS, and starts and stops tasks whenever it receives a request from ECS.
  - Container agent is only supported on Amazon EC2 instances.

AWS Fargate

- You can use Fargate with ECS to run containers without having to manage servers or clusters of EC2 instances.
- You no longer have to provision, configure, or scale clusters of virtual machines to run containers.
- Fargate only supports container images hosted on Elastic Container Registry (ECR) or Docker Hub.

Task Definitions for Fargate Launch Type

- Fargate task definitions require that the network mode is set to *awsvpc*. The *awsvpc* network mode provides each task with its own elastic network interface.
- Fargate task definitions only support the *awslogs* log driver for the log configuration. This configures your Fargate tasks to send log information to Amazon CloudWatch Logs.
- Task storage is **ephemeral**. After a Fargate task stops, the storage is deleted.

Monitoring

- You can configure your container instances to send log information to CloudWatch Logs. This enables you to view different logs from your container instances in one convenient location.
- With CloudWatch Alarms, watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods.
- Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs.

EC2 Container Services ECS vs Lambda

Amazon EC2 Container Service (ECS)

- Amazon ECS is a highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances. ECS eliminates the need for you to install, operate, and scale your own cluster management infrastructure.
- With ECS, deploying containerized applications is easily accomplished. This service fits well in running batch jobs or in a microservice architecture.
  You have a central repository where you can upload your Docker Images from ECS container for safekeeping called Amazon ECR.
- Applications in ECS can be written in a stateful or stateless matter.
- The Amazon ECS CLI supports Docker Compose, which allows you to simplify your local development experience as well as easily set up and run your containers on Amazon ECS.
- Since your applications still run on EC2 instances, server management is your responsibility. This gives you more granular control over your system.
- It is up to you to manage scaling and load balancing of your EC2 instances as well, unlike in AWS Lambda where functions scale automatically.
- You are charged for the costs incurred by your EC2 instances in your clusters. Most of the time, Amazon ECS costs more than using AWS Lambda since your active EC2 instances will be charged by the hour.
- One version of Amazon ECS, know as AWS Fargate, will fully manage your infrastructure so you can just focus on deploying containers. AWS Fargate has a different pricing model from the standard EC2 cluster.
- ECS will automatically recover unhealth containers to ensure that you have the desired number of containers supporting your application.

Tutorials Dojo

AWS Lambda

- AWS Lambda is a function-as-a-service offering that runs your code in response to events and automatically manages the compute resources for you, since Lambda is a serverless compute service. With Lambda, you do not have to worry about managing servers, and directly focus on your application code.
- Lambda automatically scales your function to meet demands. It is noteworthy, however, that Lambda has a maximum execution duration per request of 900 seconds or 15 minutes.
- To allow your Lambda function to access other services such as Cloudwatch Logs, you would need to create an execution role that has the necessary permissions to do so.
- You can easily integrate your function with different services such as API Gateway, DynamoDB, CloudFront, etc. using the Lambda console.
- You can test your function code locally in the Lambda console before launching it into production. Currently, Lambda supports only a number of programming languages such as Java, Go, PowerShell, Node.js, C#, Python, and Ruby. ECS is not limited by programming languages since it mainly caters to Docker.
- Lambda functions must be stateless since you do not have volumes for data storage.
- You are charged based on the number of requests for your functions and the duration, the time it takes for your code to execute. To minimize costs, you can throttle the number of concurrent executions running at a time, and the execution time limit of the function.
- With Lambda@Edge, AWS Lambda can run your code across AWS locations globally in response to Amazon CloudFront events, such as requests for content to or from origin servers and viewers. This makes it easier to deliver content to end users with lower latency.