AWS®
Certified Developer
Official Study Guide
Associate (DVA-C01) Exam

AWS CodeDeploy *AWS CodeDeploy* automates code deployments to any instance. It handles the complexity of updating your applications, which avoids downtime during application deployment. It deploys to Amazon EC2 or on-premises servers, in any language and on any operating system. It also integrates with third-party tools and AWS.

Deploying Highly Available and Scalable Applications

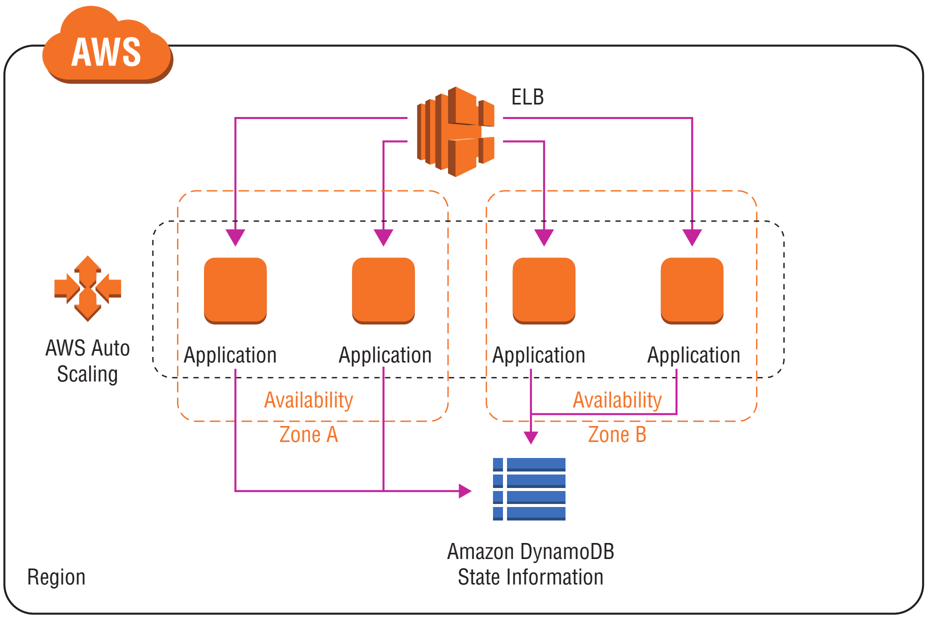
Load balancing is an integral part to directing and managing traffic among your instances. As you launch applications in your environments, you will want them to have high performance and high availability for your users. To enable both of these features, a load balancer will be necessary.

Elastic Load Balancing (ELB) supports three types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers. You select a load balancer based on your application needs.

- The *Application Load Balancer* provides advanced request routing targeted at delivery of modern application architectures, including microservices and container-based applications. It simplifies and improves the security of your application by ensuring that the latest Secure Sockets Layer (SSL)/Transport Layer Security (TLS) ciphers and protocols are used at all times. The Application Load Balancer operates at the request level (Layer 7) to route HTTP/HTTPS traffic to its targets: Amazon EC2 instances, containers, and IP addresses based on the content of the request. It is ideal for advanced load balancing of HTTP and HTTPS traffic.
- The *Network Load Balancer* operates at the connection level (Layer 4) to route TCP traffic to targets: Amazon EC2 instances, containers, and IP addresses based on IP protocol data. It is the best option for load balancing of TCP traffic because it's capable of handling millions of requests per second while maintaining ultra-low latencies. Network Load Balancer is optimized to handle sudden and volatile traffic patterns while using a single static IP address per Availability Zone. It is integrated with other popular AWS services, such as AWS Auto Scaling, Amazon Elastic Container Service (Amazon ECS), and AWS CloudFormation. Amazon ECS provides management for deployment, scheduling, and scaling, and management of containerized applications.
- The *Classic Load Balancer* provides basic load balancing across multiple Amazon EC2 instances and operates at both the request level and the connection level. The Classic Load Balancer is intended for applications that were built within the EC2-Classic network. When you're using Amazon Virtual Private Cloud (Amazon VPC), AWS recommends the Application Load Balancer for Layer 7 and Network Load Balancer for Layer 4).

Figure 6.4 displays the flow for deploying highly available and scalable applications.

FIGURE 6.4 Deploying highly available and scalable applications



The flow for deploying highly available and scalable applications includes the following components:

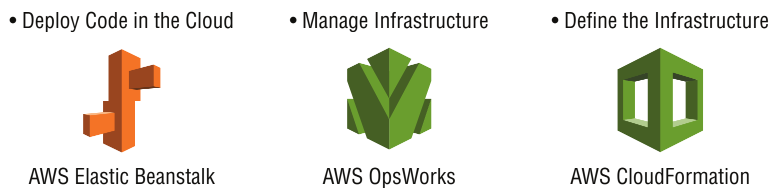
- Multiple Availability Zones and AWS Regions.
- Health check and failover mechanism.
- Stateless application that stores the session state in a cache server or database.
- AWS services that help you to achieve your goal. For example, Auto Scaling helps you maintain high availability and scalability.

TIP Elastic Load Balancing and Auto Scaling are designed to work together.

Deploying and Maintaining Applications

AWS provides several services to manage your application and resources, as shown in Figure 6.5.

FIGURE 6.5 Deployment and maintenance services



With AWS Elastic Beanstalk, you do not have to worry about managing the infrastructure for your application. You deploy your application, such as a Ruby application, in a Ruby container, and Elastic Beanstalk takes care of scaling and managing it.

AWS OpsWorks is a configuration and deployment management tool for your Chef or Puppet resource stacks. Specifically, *OpsWorks for Chef Automate* enables you to manage the lifecycle of your application in layers with Chef recipes. It provides custom Chef cookbooks for managing many different types of layers so that you can write custom Chef recipes to manage any layer that AWS does not support.

AWS CloudFormation is infrastructure as code. The service helps you model and set up AWS resources so that you can spend less time managing them. It is a template-based tool, with formatted text files in JSON or YAML. You can create templates to define what AWS infrastructure you want to build and any relationships that exist among the parts of your AWS infrastructure.

NOTE

Use AWS CloudFormation templates to provision and configure your stack resources.

Automatically Adjust Capacity

Use AWS Auto Scaling to monitor the AWS resources that are part of your application. The service automatically adjusts capacity to maintain steady, predictable performance. You can build scaling plans to manage your resources, including Amazon EC2 instances and Spot Fleets, Amazon Elastic Container Registry (Amazon ECR) tasks, Amazon DynamoDB tables and indexes, and Amazon Aurora Replicas.

AWS Auto Scaling makes scaling simple, with recommendations that allow you to optimize performance, costs, or balance between them. If you are already using EC2 Auto Scaling to scale your Amazon EC2 instances dynamically, you can now combine it with AWS Auto Scaling to scale additional resources for other AWS services. With AWS Auto Scaling, your applications have the right resources at the right time.

Auto Scaling Groups

An Auto Scaling group contains a collection of Amazon EC2 instances that share similar characteristics. This collection is treated as a logical grouping to manage the scaling of instances. For example, if a single application operates across multiple instances, you might want to increase the number of instances in that group to improve the performance of the application or decrease the number of instances to reduce costs when demand is low.

You can use the Auto Scaling group to scale the number of instances automatically based on criteria that you specify or maintain a fixed number of instances even if an instance becomes unhealthy. This automatic scaling and maintaining the number of instances in an Auto Scaling group make up the core functionality of the EC2 Auto Scaling service.

An Auto Scaling group launches enough Amazon EC2 instances to meet its desired capacity. The Auto Scaling group maintains this number of instances by performing periodic health checks on the instances in the group. If an instance becomes unhealthy, the group terminates the unhealthy instance and launches another instance to replace it.

You can use scaling policies to increase or decrease the number of instances in your group dynamically to meet changing conditions. When the scaling policy is in effect, the Auto Scaling group adjusts the desired capacity of the group and launches or terminates the instances as needed. You can also manually scale or scale on a schedule.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is an AWS service that you can use to deploy applications, services, and architecture. It provides provisioned scalability, load balancing, and high availability. It uses common languages, including Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker, on common-type web servers, such as Apache, NGINX, Passenger, and IIS.

NOTE Elastic Beanstalk charges only for the resources you use to run your application.

Elastic Beanstalk is a solution that enables the automated deployments and management of applications on the AWS Cloud. Elastic Beanstalk can launch AWS resources automatically with Amazon Route 53, AWS Auto Scaling, Elastic Load Balancing, Amazon EC2, and Amazon Relational Database Service (Amazon RDS) instances, and it allows you to customize additional AWS resources.

Deploy applications without worrying about managing the underlying technologies, including the following:

Components

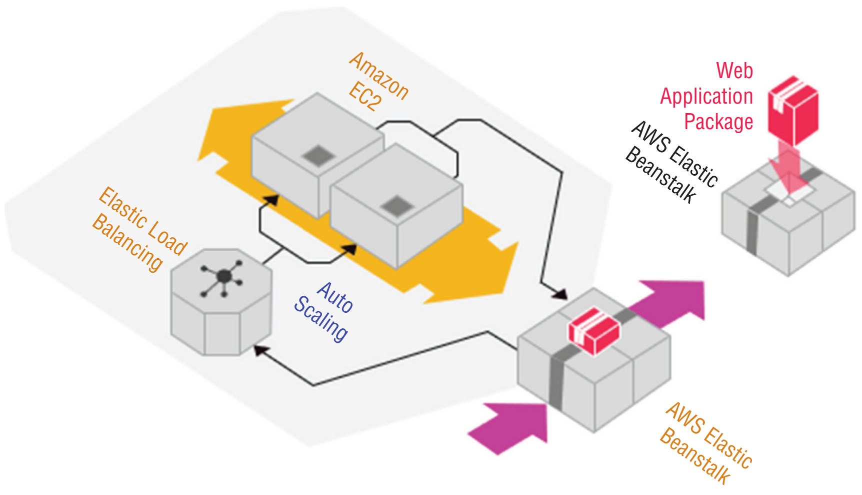
- Environments
- Application versions
- Environment configurations

Permission Model

- Service role
- Instance profile

Figure 6.6 displays the Elastic Beanstalk underlying technologies.

FIGURE 6.6 AWS Elastic Beanstalk underlying technologies



Elastic Beanstalk supports customization and N-tier architectures. It mitigates common manual configurations required in a traditional infrastructure deployment model. With Elastic Beanstalk, you can also create repeatable environments and reduce redundancy, thus rapidly updating environments and facilitating service-managed application stacks. You can deploy multiple environments in minutes and use various automated deployment strategies.

TIP AWS Elastic Beanstalk allows you to focus on building your application.

Implementation Responsibilities

AWS and our customers share responsibility for achieving a high level of software component security and compliance. This shared model reduces your operational burden. The service you select determines the level of your responsibility. For example, Elastic Beanstalk helps you perform your side of the shared responsibility model by providing a managed updates feature. This feature automatically applies patch and minor updates for an Elastic Beanstalk supported platform version.

Developer Teams

Using AWS Elastic Beanstalk, you build full-stack environments for web and worker tiers. The service provides a preconfigured infrastructure.

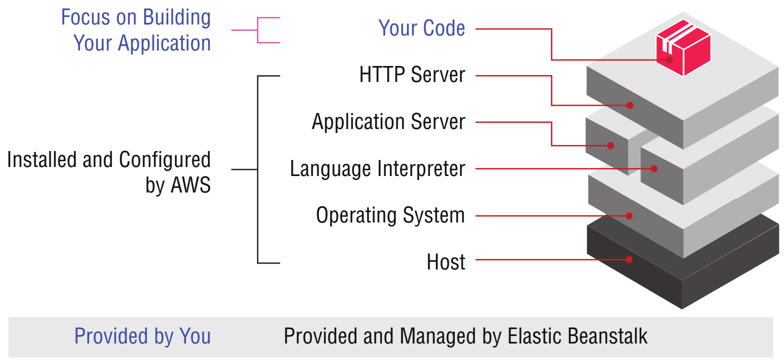
- Single-instance (development, low cost)
- Load balanced, AWS Auto Scaling (production)

Elastic Beanstalk Responsibilities

Elastic Beanstalk provisions the necessary infrastructure resources, such as the load balancer, Auto Scaling group, security groups, and database (optional). It also provides a unique domain name for your application (for example, `yourapp.elasticbeanstalk.com`).

Figure 6.7 displays Elastic Beanstalk responsibilities.

FIGURE 6.7 AWS Elastic Beanstalk responsibilities


Focus on Building Your Application
Your Code
HTTP Server
Application Server
Installed and Configured by AWS
Language Interpreter
Operating System
Host
Provided by You
Provided and Managed by Elastic Beanstalk

Working with Your Source Repository

Developer teams generally begin their SDLC processes by managing their source code in a source repository. Uploading and managing the multiple changes on application source code is a repeated process. With Elastic Beanstalk, you can create an application, upload a version of the application as a source bundle, and provide pertinent information about the application.

The first step is to integrate Elastic Beanstalk with your source code to create your source bundle. As your source repository, you can install Git for your applications or use an existing repository and map your current branch from a local repository in Git to retrieve the source code.

Alternatively, you can use AWS CodeCommit as a source control system to retrieve source code. By using Elastic Beanstalk with the AWS CodeCommit repository, you extract from a current branch on CodeCommit.

To deploy a new application or application version, Elastic Beanstalk works with source bundles or packaged code. Prepare the code package with all of the necessary code dependencies and components.

Elastic Beanstalk can either retrieve the source bundle from a source repository or download the bundle from an Amazon Simple Storage Service (Amazon S3) bucket. You can use the IAM role to grant Elastic Beanstalk access to all services. The service accesses the source bundle from the location you designate, extracts the components from the bundle, deploys new application versions by launching the code, creates and configures the infrastructure, and allocates the platform on Amazon EC2 instances to run the code.

The application runs on the resources and instances that the service generates. Your configuration for these resources and your application will become your environment settings, supporting the entire configuration of your deployment. Each deployment has an auto-incremented deployment identity (ID), so you are able to manage your multiple running deployments. Think of these as multiple running code releases in the AWS Cloud.

NOTE You can also work with different hosting services, such as GitHub or Bitbucket, with your code source.

Concepts

AWS Elastic Beanstalk enables you to manage all the resources that run your application as environments. This section describes some key Elastic Beanstalk concepts.

Application

Elastic Beanstalk focuses on managing your applications as environments and all of the resources to run them. Each application that launches in the service is a logical collection of environment variables and components, application versions, and environment configurations.

Application Versions

Application versions are iterations of the application's deployable code. Application versions in Elastic Beanstalk point to an Amazon S3 object with the code source package. An application can have many versions, with each version being unique. You can deploy and access any application version at any time. For example, you may want to deploy different versions for different types of tests.

Environment

Each Elastic Beanstalk environment is a separate version of the application, and that version's AWS Cloud components deploy onto AWS resources to support that version. Each environment runs one application version at a time, but you can run multiple environments, with the same application on each, along with its own customizations and resources.

Environment Tier

To launch an environment, you must first choose an environment tier. Elastic Beanstalk provisions the required resources to support both the infrastructure and types of requests the application will support. The environment can launch and access other AWS resources. For example, it may pull tasks from Amazon Simple Queue Service (Amazon SQS) queues or store temporary configuration files in Amazon S3 buckets (according to your customizations). Each environment will then have an environment configuration—a collection of settings and parameters based on your customizations that define associated resources and how the environment will work.

Environment Configuration

You can change your environment to create, modify, delete, or deploy resources and change the settings for each. Your environment configuration saves to a configuration template exclusive to each environment and is accessible by either the Elastic Beanstalk application programming interface (API) calls or the service's command line interface (EB CLI).

In Elastic Beanstalk, you can run either a web server environment or a worker environment. Figure 6.8 displays an example of a web server environment running in Elastic Beanstalk with Amazon Route 53 as the domain name service (DNS) and ELB to route traffic to the web server instances.

FIGURE 6.8 Application running on AWS Elastic Beanstalk

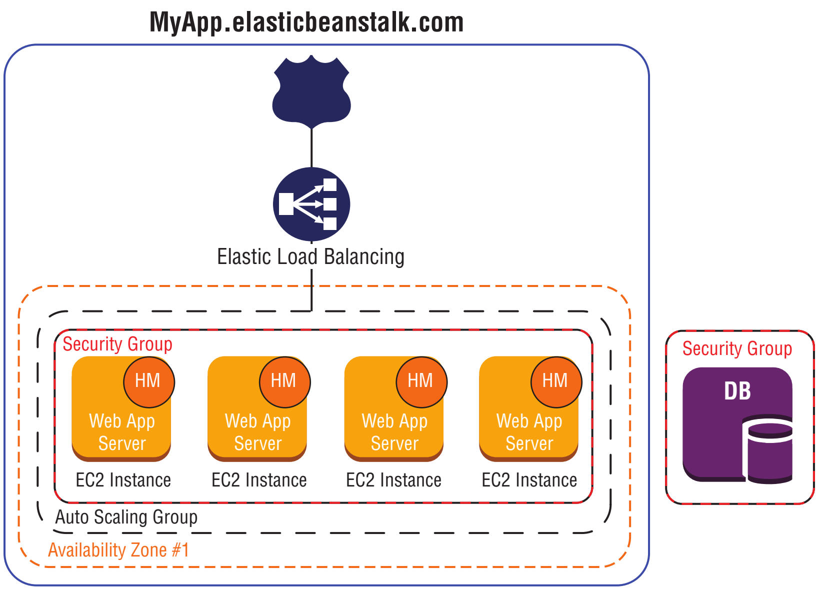
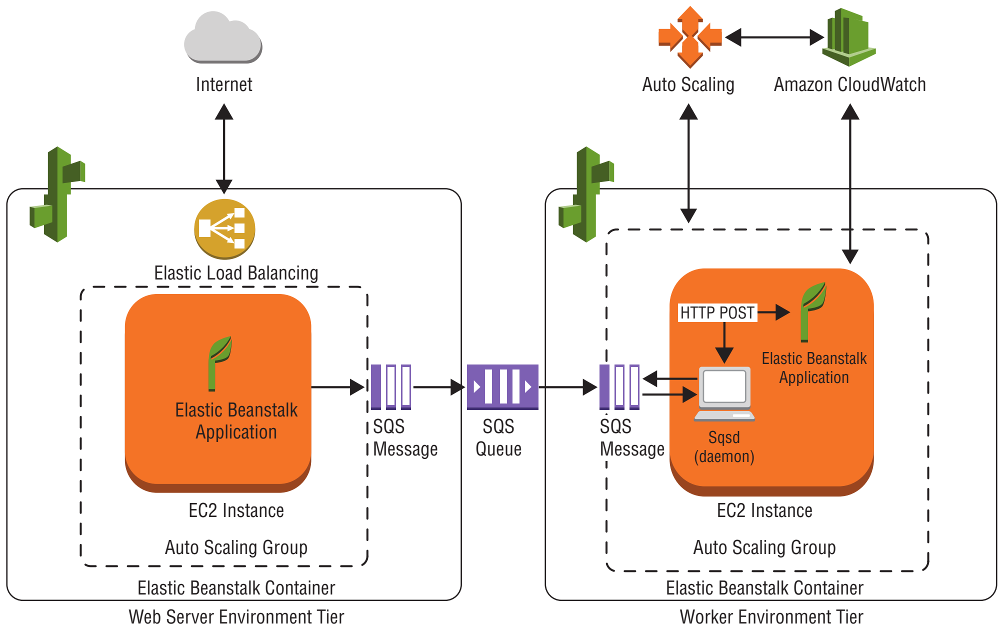


Figure 6.9 shows a worker environment architecture, where AWS resources create configurations, such as Auto Scaling groups, Amazon EC2 instances, and an IAM role, to manage resources for your worker applications.

FIGURE 6.9 Worker tier on AWS Elastic Beanstalk



For the worker environment tier, Elastic Beanstalk creates and provisions additional resources and files to support the tier. This includes services like Amazon SQS queues operating between worker applications, AWS Auto Scaling groups, security groups, and EC2 instances.

The worker environment infrastructure uses all of your customization and provision resources to determine the types of requests it receives.

Docker Containers

You can also use Docker containers with Elastic Beanstalk to run your applications from a container. Install Docker, choose the software you require, and select the Docker images you want to launch. Define your runtime environment, platform, programming language, and application dependencies and tools. Docker containers are self-contained and include configurations and software that you specify for your application to run. Each Docker container restarts automatically if another container crashes. When you choose to deploy your applications with Docker containers, your infrastructure is provisioned with capacity provisioning, load balancing, scaling, and health monitoring, much like a noncontainer environment. You can continue to manage your application and the AWS resources you use.

Docker requires platform configurations that enable you to launch single or multicontainer deployments. A single container deployment launches a single Docker image, and your application uses a single container configuration for a single Amazon EC2 instance.

A multicontainer deployment uses the Amazon ECS to launch a cluster of containers with Docker images. A multicontainer configuration is applied to each instance. You can also run preconfigured Docker platform configurations with generic customization for popular software stacks that you want to use for your application.

AWS Elastic Beanstalk Command Line Interface

Elastic Beanstalk has its own command line interface separate from the AWS CLI tool. To create deployments from the command line, you download and install the AWS Elastic Beanstalk CLI (EB CLI).

Table 6.1 lists common EB CLI commands.

TABLE 6.1 Common AWS Elastic Beanstalk Commands

Command	Definition
`eb init application-name`	Sets default values for Elastic Beanstalk applications with the EB CLI configuration wizard
`eb create`	Creates a new environment and deploys an application version to it
`eb deploy`	Deploys the application source bundle from the initialized project directory to the running application
`eb clone`	Clones an environment to a new environment so that both have identical environment settings
`eb codesource`	Configures the EB CLI to deploy from an AWS CodeCommit repository, or disables AWS CodeCommit integration and uploads the source bundle from your local machine

Customizing Environment Configurations

You can use Elastic Beanstalk to customize the platforms used to support your application and your infrastructure. To do so, create a configuration file in the `ebextensions` directory (or `.ebextensions`) to include with your web application's source code. The configuration file allows for simple and advanced customizations of your environment and contains settings for your AWS resources. To deploy customized resources to support your application source bundle, use YAML to configure the file.

The configuration file has several sections. The `option_settings` section defines your configuration option values for your AWS resources. The `resources` section adds further customization in your application environment beyond the service functionality, which includes AWS CloudFormation–supported resources that Elastic Beanstalk can access and run. The remaining sections allow for fine-grained configurations to integrate packages, sources, files, and container commands.

NOTE Launch environments from integrated development environment (IDE) tools to avoid poorly formatted configurations and source bundles that could cause unrecoverable failures.

You apply configuration files in the `ebextensions` directory to Elastic Beanstalk stacks. The stacks are the AWS resources that you allocate for your infrastructure and application. If you have any resource, such as Amazon VPC, Amazon EC2, or Amazon S3, that was updated or configured, these files deploy with your changes. You can zip your `ebextension` files, upload, and apply them to multiple application environments. You can view your environment variables in `option_settings` for future evaluation or changes. These are accessible from the AWS Management Console, command line, and API calls.

TIP You can view Elastic Beanstalk stacks in AWS CloudFormation, but always use the Elastic Beanstalk service and `ebextensions` to make modifications. This way, edits and modifications to the application stacks are simplified without introducing unrecoverable failures.

Elastic Beanstalk generates logs that you can view to troubleshoot your environments and resources. The logs display Amazon EC2 operational logs and logs that are specific to servers running for your applications.

Integrating with Other AWS Services

Elastic Beanstalk automatically integrates or manages other AWS services with application code to provision efficient working environments. However, you might find it necessary to add additional services, such as Amazon S3 for content storage or Amazon DynamoDB for data records, to work with an environment. To grant access between any integrated service and Elastic Beanstalk, you must configure permissions in IAM.

Amazon S3

You can use Amazon S3 to store static content you want to integrate with your application and point directly to objects you store in Amazon S3 from your application or from other resources. In addition to setting permissions in IAM policies, take advantage of presigned URLs for controlled Amazon S3 `GET` and `PUT` operations.

Amazon CloudFront

You can integrate your Elastic Beanstalk environment with Amazon CloudFront, which provides content delivery and distribution through the use of edge locations throughout the world. This can decrease the time in which your content is delivered to you, as the content is cached and routed through the closest edge location serving you. After you deploy your application on Elastic Beanstalk, use the Amazon CloudFront content delivery network (CDN) to cache static content from your application. To identify the source of your content in Amazon CloudFront, you can use URL path patterns to cache your content and then retrieve it from the cache. This approach serves your content more rapidly and offloads requests directly sourced from your application.

AWS Config

With *AWS Config*, you can visualize configuration history and how configurations evolve over time. Tracking changes helps you to fulfill compliance obligations and meet auditing requirements. You can integrate AWS Config directly with your application and its versions or your Elastic Beanstalk environment. *You can customize AWS Config to record changes per resource, per region, or globally.* In the AWS Config console, you can select Elastic Beanstalk resource types to record specific applications and environment resources. You can view the recorded information in the AWS Config dashboard under Resource Inventory.

Amazon RDS

Various options are available for creating databases for your environment, such as Amazon Relational Database Service (Amazon RDS) for SQL databases and Amazon DynamoDB for NoSQL databases. Elastic Beanstalk can create a database and store and retrieve data for any of your environments. Each service has its own features to handle scaling, capacity, performance, and availability.

To store, read, or write to your data records, you can set up an Amazon RDS database instance or an Amazon DynamoDB table by using the same configuration files for your other service option settings. You must create connections to the database, which require you to set up password management in Elastic Beanstalk. Your configurations are saved in the `ebextensions` directory. You can also create direct connections, within your application code or application configuration files, to both internal and external databases. When using Amazon RDS, avoid accidentally deleting and re-creating databases without a properly installed backup. To reduce the risk of losing data, take a manual snapshot of the master Amazon RDS database immediately before deleting.

NOTE If you create periodic tasks with a worker environment, Elastic Beanstalk automatically creates an Amazon DynamoDB table to perform leader election and stores task information.

Amazon ElastiCache

For caching capabilities, you can integrate Amazon ElastiCache service clusters with the Elastic Beanstalk environment. If you use a nonlegacy container, you can set your configuration files to use the supported container and then offload requests to the cache cluster.

Doing so enables you to increase the performance of your application and databases running in your Elastic Beanstalk environment.

AWS Identity and Access Management Roles

Elastic Beanstalk integrates with AWS Identity and Access Management (IAM) roles to enable access to the services you require to run your architecture.

When you launch the service to create an environment, a default service role and instance profile are created for you through the service API. Managed policies for resources permissions are also attached, including policies for Elastic Beanstalk instance health monitoring within your infrastructure and platform updates that can be made on behalf of the service. These policies, called `AWSElasticBeanstalkEnhancedHealth` and `AWSElasticBeanstalkService`, attach to the default service role and enable the default service role to specify a trusted entity and trust policy.

When you use commands from the EB CLI, the role allows automatic management of the AWS Cloud that services you run. The service creates an environment, if you don't identify it specifically; creates a service-linked role; and uses it when you spin up a new environment. To create the environment successfully, the `CreateServiceLinkedRole` policy must be available in your IAM account.

You use IAM roles to automate the management of allocated services for your application through Elastic Beanstalk. With IAM, you can also launch code with inline policies. It is important to understand how the service creates and uses the roles to keep your application and data secure.

TIP For IAM to manage the policies for the account better, create policies at the account level.

Deployment Strategies

A *deployment* is the process of copying content and executing scripts on instances in your deployment group. To accomplish this, AWS CodeDeploy performs the tasks outlined in the `AppSpec` configuration file. For both Amazon EC2 on-premises instances and AWS Lambda functions, the deployment succeeds or fails based on whether individual `AppSpec` tasks complete successfully.

After you have created a deployment, you can update it as your application or service changes. You can update a deployment by adding or removing resources from a deployment, thus updating the properties of existing resources in a deployment.

NOTE A serverless application is typically a combination of AWS Lambda and other AWS services.

To create seamless deployments, choose an effective deployment strategy. Each strategy has specific advantages relative to different use cases. Appropriate strategies help create deployments where you experience minimal or no downtime, and you can apply the strategy for different purposes within your environments. Each change needs a strategy that best fits your application deployments.

All-at-Once and In-Place Deployments

An *all-at-once deployment* applies updates to all your instances at once. When you execute this strategy, you experience downtime, as all instances receive the change at the same time.

This is an appropriate strategy for simple, immediate update requirements when it's not critical to have your application always available, and you're comfortable with the site being offline for a short duration. To enable all-at-once updates, set a deployment policy either in the AWS Management Console or in the command line (`DeploymentPolicy`).

When you perform an *in-place deployment*, AWS CodeDeploy stops currently running applications on the target instance, deploys the latest revision, restarts applications, and validates successful deployment. In-place deployments can support the automatic configuration of a load balancer. In this case, the instance is deregistered from the load balancer before deployment and registered again after the deployment processes successfully.

In-place updates are also available for your platform updates, such as a coding-language platform update for a web server. Select the new platform and then run the update from the AWS Management Console or command line directly as a platform update.

NOTE AWS Lambda does not support in-place deployments.

Rolling Deployments

A *rolling deployment* applies changes to all of your instances by rolling the updates from one instance to another. Elastic Beanstalk can deploy configuration changes in batches. This approach reduces possible downtime during implementation of the change and allows available instances to run while you deploy.

As updates are applied in a batch, the batch will be out of service for a short period while the changes propagate and then relaunch with the new configuration. When the change is complete, the service moves on to the next batch of instances to apply the changes. With this strategy, you can implement both periodic changes and pauses between updates. For example, you might specify a time to wait between health-based updates so that instances must pass health checks before moving on to the next batch. If the rolling update fails, the service begins another rolling update for a rollback to the previous configuration.

Rolling updates include changes for Auto Scaling group configurations, Amazon EC2 instance configurations, and Amazon VPC settings. It is an effective method for updating an application version on fleets of instances through the Elastic Beanstalk service. To enable rolling updates, set a deployment policy either in the AWS Management Console or in the command line (`DeploymentPolicy`) and choose this strategy along with specific options. You can select *Rolling* or *Rolling with additional batch*. By using *Rolling with additional batch*, you can launch a new batch of instances before you begin to take instances out of service for your rolling updates. This option provides an available batch for rollback from a failed update. After the deployment is successfully executed, Elastic Beanstalk terminates the instances from the additional batch. This is helpful for a critical application that must continue running with less downtime than the standard rolling update.

Blue/Green Deployment

When high availability is critical for applications, you may want to choose a *blue/green deployment*, where your newer environment will be separate from your existing environment. The running production environment is considered the *blue environment*, and the newer environment with your update is considered the *green environment*. When your changes are ready and have gone through all tests in your green environment, you can swap the CNAMEs of the environments to redirect traffic to the newer running environment. This strategy provides an instantaneous update with typically zero downtime.

When you deploy to AWS Lambda functions, blue/green deployments publish new versions of each function. Traffic shifting then routes requests to the new functioning versions according to the deployment configuration you define.

If your infrastructure contains Amazon RDS database instances, the data does not automatically transfer to the new environment. Without performing backups, you will experience data loss when you use the blue/green strategy. If you have Amazon RDS instances in your infrastructure, implement a different deployment strategy or a series of steps to create snapshot backups outside of Elastic Beanstalk before you execute this type of deployment.

Immutable Deployment

An *immutable deployment* is best when an environment requires a total replacement of instances, rather than updates to an existing part of an infrastructure. This approach implements a safety feature for updates and rollbacks. Elastic Beanstalk creates a temporary Auto Scaling group behind your environment's load balancer to contain the new instances with the updates you apply. If the update fails, the rollback process terminates the Auto Scaling group. Immutable instances implement a number of health checks. If all instances pass these checks, Elastic Beanstalk transfers the new configurations to the original Auto Scaling group, providing an additional check before you apply your changes to other instances. Enhanced health reports evaluate instance health in the update. After the updates are made, Elastic Beanstalk deletes the temporary Auto Scaling group of the older instances.

NOTE

During this type of deployment, your capacity doubles for a short duration between the updates and terminations of instances. Before you use this strategy, verify that your instances have a low on-demand limit and enough capacity to support immutable updates.

See Table 6.2 for feature comparisons between all deployment strategies. The check mark indicates options that the deployment strategy supports.

TABLE 6.2 Deployment Strategies

Method	Impact of Failed Deployment	Deploy Time	Zero Downtime	No DNS Change	Rollback Process	Code Deployed To
All-at-once	Downtime	🕒		✓	Redeploy	Existing instances
In-place	Downtime	🕒		✓	Redeploy	Existing instances
Rolling	Single batch out of service; any successful batches before failure running new application version	🕒🕒	✓	✓	Redeploy	Existing instances
Rolling with additional batch	Minimal if first batch fails; otherwise, similar to Rolling	🕒🕒🕒	✓	✓	Redeploy	New and existing instances
Blue/Green	Minimal	🕒🕒🕒🕒	✓		Swap URL	New instances
Immutable	Minimal	🕒🕒🕒🕒	✓	✓	Redeploy	New instances

Container Deployments

Elastic Beanstalk enables you to launch your applications with Docker containers. With a Docker container, you can create a runtime environment with all of the dependencies, packages, and tools that your application may require to run. Your container can have all of the configurations necessary for your application. By using Docker with Elastic Beanstalk, you have the infrastructure for capacity provisioning, scalability, load balancing, and health monitoring for the instances that run on containers. The containers integrate with your

Amazon VPC for network requirements and with IAM to enable resource management. You can launch different software engines with containers to provide various options and third-party tools to run containers.

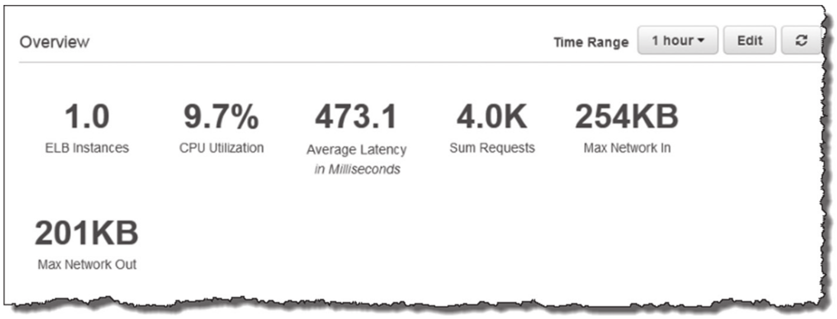
You can choose from single container configurations and multicontainer configurations. A single container runs one container per instance. A multicontainer runs multiple applications or engines on one instance, with all of the software and settings you require. Preconfigured options are available with Docker, and you can integrate them with instances that run in your architecture through Elastic Beanstalk.

Monitoring and Troubleshooting

After you launch your code, check on its performance and availability. You can monitor statistics and view information about the health of your application, its environment, and specific services from the AWS Management Console. Elastic Beanstalk also creates alerts that trigger at established thresholds to monitor your environment's health. In the AWS Management Console, the AWS Elastic Beanstalk Monitoring page shows aggregated statistics and graphs for your applications and resources. Each environment is color-coded to indicate the environment's status. You can see at a glance whether your environment is available online at any point in time. Metrics gathered by the resources in your environment are published to Amazon CloudWatch in five-minute intervals. You can adjust the time range for the statistics and graphs and customize your views of the metrics.

Figure 6.10 shows an example of the statistics that you can view for your environment.

FIGURE 6.10 Health dashboard on AWS Elastic Beanstalk



1.0 ELB Instances
9.7% CPU Utilization
473.1 Average Latency *in Milliseconds*
4.0K Sum Requests
254KB Max Network In
201KB Max Network Out

Figure 6.11 shows an example of the graphs that you can view.

FIGURE 6.11 Metrics for monitoring on AWS Elastic Beanstalk

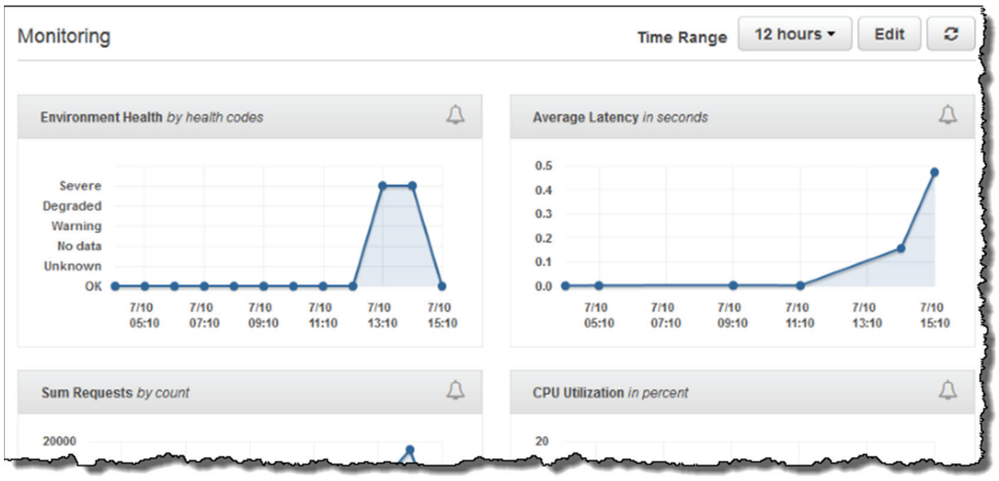


Table 6.3 defines the AWS Elastic Beanstalk Monitoring page colors.

TABLE 6.3 AWS Elastic Beanstalk Health Page Color Definitions

Color	Description
Gray	Your environment is being updated.
Green	Your environment has passed the most recent health check. At least one instance in your environment is available and taking requests.
Yellow	Your environment has failed one or more health checks. Requests to your environment are failing.
Red	Your environment has failed three or more health checks, or an environment resource has become unavailable. Requests are consistently failing.

By default, Elastic Beanstalk displays Amazon EC2, Auto Scaling, and Elastic Load Balancing metrics for your application environments. These metrics are available to you on your AWS Elastic Beanstalk Monitoring page as soon as you deploy your application environment. You can access the health status from the AWS Management Console or the EB CLI.

Basic Health Monitoring

To access the health status from the AWS Management Console, select the Elastic Beanstalk service and then select the tab for your specific application environment. An environment overview shows your architecture's instance status details, resource details, and filter capabilities. Health statuses are indicated in four distinct colors.

To access the health status from the EB CLI, enter the `eb health` command. The output shows the environment and the health of associated instances. Enhanced health reporting also provides the following seven health statuses, which are single-word descriptors that provide a better indication of the state of your environment:

```
ok    warning    degraded    severe    info    pending    unknown
```

You can also use the `eb status` command in the EB CLI or the `DescribeEnvironments` API call to retrieve the health status for an environment. You can check the health of the overall environment or the individual services of Amazon EC2 or an Elastic Load Balancing load balancer. Health checks on your Elastic Load Balancing port execute both for the default port 80 and a custom Elastic Load Balancing port/path.

For `GET` requests with the load balancer, `200 OK` is the default success code and indicates a healthy status. The service can also return `400` level responses. You can also configure a health check URL for custom static page responses.

NOTE Be sure to adjust the caching time to live for any health check static pages or URLs in Amazon CloudFront or for any caching mechanism you may use.

Elastic Beanstalk also reports missing configurations or other issues that could affect the health of the application environment.

Enhanced Health Monitoring

There are two types of reporting: the default health information about your resources and the enhanced health reporting that provides you more information for monitoring health.

You can use the enhanced health reporting feature to gather additional resource data and display graphs and statistics of environment health in greater detail. This is important when you deploy multiple versions of your application and when you need to analyze factors that could be degrading your application's availability or performance. You can view these details in the AWS Elastic Beanstalk Monitoring page from the AWS Management Console. These reports require the creation of two IAM roles: a *service role* to allow access between the services and Elastic Beanstalk and an *instance profile* to write logs into an Amazon S3 bucket.

NOTE Running the enhanced health report requires a version 2 or newer platform configuration that supports all platforms except Windows Server with IIS. The enhanced health reports provide data directly to Elastic Beanstalk and do not run through Amazon CloudWatch.

When you package dependencies for multiple cookbooks in the parent directory of the cookbooks, create a `Berksfile` such as this:

```
source "https://supermarket.chef.io"
cookbook "server-app", path: "./server-app"
cookbook "server-utils", path: "./server-utils"
```

After you package the dependencies, run the `berks package` command from this directory to download and dependencies for your cookbooks.

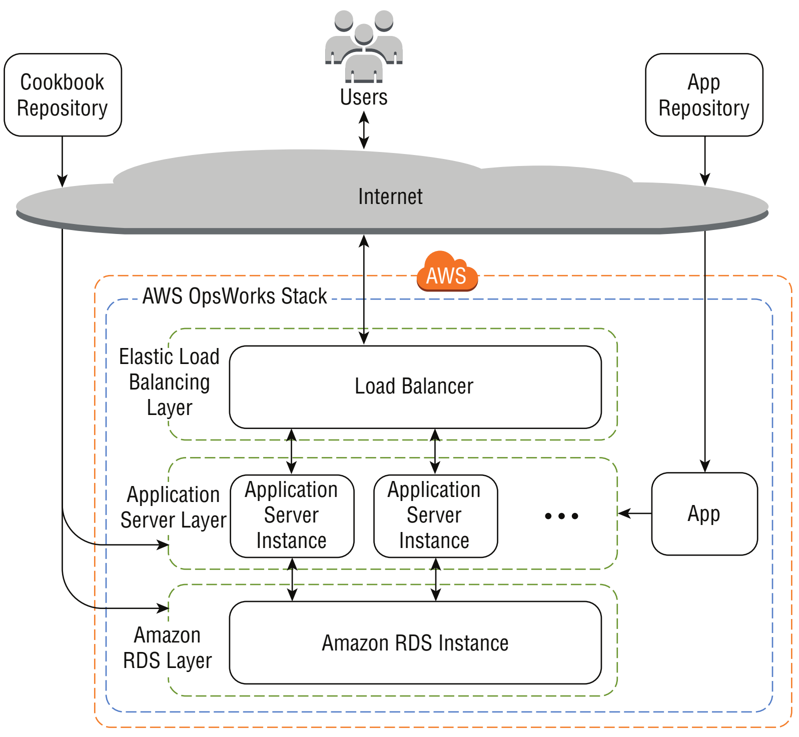
```
berks package cookbooks.tar.gz
```

Stack

A typical workload in AWS will include systems for various purposes, such as load balancers, application servers, proxy servers, databases, and more. The set of Amazon EC2 on-premises instances, Amazon RDS, Elastic Load Balancing, and other systems make up a stack. You can organize stacks across an enterprise.

Suppose you have a single application with dev, test, and production environments. Each of these environments has a stack that enables you to separate resources to ensure stability of changes. You group resources into stacks by logical or functional purposes. The example stack in Figure 9.4 includes three layers, a cookbook repository, an application repository, and one app to deploy to the application server instances. This stack manages a full application available to users over the Internet.

FIGURE 9.4 Example stack structure



When you create a new stack, you will have the option to set the stack's properties.

Stack Name

Stack Name identifies stacks in the AWS OpsWorks Stacks console. Since this name is not unique, AWS OpsWorks assigns a Globally Unique Identifier (GUID) to the stack after you create it.

API Endpoint Region

AWS OpsWorks associates a stack with either a global endpoint or one of multiple regional endpoints. When you create a resource in the stack, such as an instance, it is available only from the endpoint you specify when you create the stack. For example, if a stack is created with the global "classic" endpoint, any instances will be accessible only by AWS OpsWorks Stacks that use the global API endpoint in the US East (N. Virginia) region. Resources are not available across regional endpoints.

Amazon Virtual Private Cloud

Stacks can create and manage instances in Amazon EC2 Classic or an Amazon Virtual Private Cloud (Amazon VPC). When you select an Amazon VPC, you will be able to specify in what subnets to deploy instances when they are created.

Default Operating System

AWS OpsWorks Stacks supports many built-in Linux operating systems and Windows Server (only in Chef 12.2 stacks). If there is a custom Amazon Machine Images (AMI) you want to use, you must configure other tasks on the AMI to make it compatible with AWS OpsWorks Stacks. You must base the custom AMI off an AMI that AWS OpsWorks supports.

- The AMI must support `cloud-init`.
- The AMI must support the instance types you plan to launch.
- The AMI must utilize a 64-bit operating system.

Layer

A *layer* acts as a subset of instances or resources in a stack. Layers act as groups of instances or resources based on a common function. This is especially important, as the Chef recipe code applies to a layer and all instances in a layer. A layer is the point where any configuration of nodes will be set, such as what Chef recipes to execute at each lifecycle hook. A layer can contain any one or more nodes, and a node must be a member of one or more layers. When a node is a member of multiple layers, it will run any recipes you configure for each lifecycle event for both layers in the layer and recipe order you specify.

From the point of view of a Chef Server installation, a layer is synonymous with a Chef Role. In the `node` object, the layer and role data are equivalent. This is primarily to ensure compatibility with open-source cookbooks that are not written specifically for AWS OpsWorks Stacks.

Elastic Load Balancing

After a layer is created, any elastic load balancers in the same region associate with the layer. Any instances that come online in the layer will automatically register with the load balancer. The instances will also deregister from the load balancer when they go offline.

Elastic IP Addresses

You can configure layers to assign public or elastic IP addresses to instances when they come online. For Amazon Elastic Block Store (Amazon EBS) backed instances, the IP address will remain assigned after the instance stops and starts again. For instance-store backed instances, the IP address may not be the same as the original AWS OpsWorks instance.

Amazon EBS Volumes

Linux stacks include the option to assign one or more Amazon EBS volumes to a layer. In the process, you configure the mount point, size, Redundant Array of Independent Disks (RAID) configuration, volume type, Input/Output Operations Per Second (IOPS), and encryption settings. When new instances start in the layer, AWS OpsWorks Stacks will attempt to create an Amazon EBS volume with the configuration and attach it to the instance. Through the instance's setup lifecycle event, AWS OpsWorks Stacks runs a Chef cookbook to mount the volume to the instance. When the volumes add or remove volumes to or from a layer, only new instances will receive the configuration updates. Existing instances' volumes do not change.

NOTE Only Chef 11.10 stacks support RAID configurations.

Amazon RDS Layer

Amazon RDS layers pass connection information to an existing Amazon RDS instance. When you associate an Amazon RDS instance to a stack, it is assigned to an app. This passes the connection information to the instances via the app's `deploy` attributes, and you can access the data within your Chef recipes with `node[:deploy][:app_name][:database] hash`.

You can associate a single Amazon RDS instance with multiple apps in the same stack. However, you cannot associate multiple Amazon RDS instances with the same app. If your application needs to connect to multiple databases, use custom JSON to include the connection information for the other database(s).

Amazon ECS Cluster Layer

Amazon ECS cluster layers provide configuration management capabilities to Linux instances in your Amazon ECS cluster. You can associate a single cluster with a single stack at a time. To create this layer, you must register the cluster with the stack. After this, it will appear in the Layer type drop-down list of available clusters from which to create a layer, as shown in Figure 9.5.

Statistics can be used to gain insight into the health of your application and to help you determine the correct settings for various configurations. For example, you may want to implement automatic scaling on your fleet of Amazon EC2 instances in order to avoid having to launch and terminate instances manually. To do so, you must configure an Auto Scaling group. Configuration settings for an Auto Scaling group include the minimum, desired, and maximum number of instances to run in your account. By monitoring statistics over time, you can determine the minimum and maximum number of instances needed to support the average, minimum, and maximum workload.

CloudWatch statistics provide a powerful way to process large amounts of metrics at scale and present insightful data that is easy to consume. Now that you understand how CloudWatch metrics work and are organized, explore the metrics available.

Aggregations

CloudWatch aggregates metrics according to the period of time you specify when retrieving statistics. When you request this statistic, you also can have CloudWatch filter the data points based on the dimensions of the metrics. For example, in Amazon DynamoDB, metrics are fetched across all DynamoDB operations. You can specify a filter on the dimension `operations` to exclude specific operations, such as `GetItem` requests. CloudWatch does not aggregate data across regions.

Available Metrics

Table 15.1 describes the available metrics for Elastic Load Balancing resources. To discover all of the available metrics, refer to the AWS documentation.

TABLE 15.1 Elastic Load Balancing Metrics

Namespace	`AWS/ELB`
	`AWS/ApplicationELB`
	`AWS/NetworkELB`

Dimensions	`LoadBalancerName`: name of the load balancer
Key metrics	▪ `HealthyHostCount: number of responding backend servers` ▪ `RequestCount`: number of IPv4 and IPv6 requests ▪ `ActiveConnectionCount`: total number of concurrent active connections from clients

Table 15.2 describes the available Amazon EC2 metrics.

applications that have stable demand patterns and for ones that experience hourly, daily, or weekly variability in usage. AWS Auto Scaling is useful for applications that show steady demand patterns and that experience frequent variations in usage.

Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling helps you scale your Amazon EC2 instances and Spot Fleet capacity up or down automatically according to conditions that you define. AWS Auto Scaling is generally used with Elastic Load Balancing to distribute incoming application traffic across multiple Amazon EC2 instances in an AWS Auto Scaling group. AWS Auto Scaling is triggered using scaling plans that include policies that define how to scale (manual, schedule, and demand spikes) and the metrics and alarms to monitor in Amazon CloudWatch.

CloudWatch metrics are used to trigger the scaling event. These metrics can be standard Amazon EC2 metrics, such as CPU utilization, network throughput, Elastic Load Balancing observed request and response latency, and even custom metrics that might originate from application code on your Amazon EC2 instances.

You can use Amazon EC2 Auto Scaling to increase the number of Amazon EC2 instances automatically during demand spikes to maintain performance and decrease capacity during lulls to reduce costs.

Dynamic Scaling

The *dynamic scaling* capabilities of Amazon EC2 Auto Scaling refers to the functionality that automatically increases or decreases capacity based on load or other metrics. For example, if your CPU spikes above 80 percent (and you have an alarm set up), Amazon EC2 Auto Scaling can add a new instance dynamically, reducing the need to provision Amazon EC2 capacity manually in advance. Alternatively, you could set a target value by using the new `Request Count Per Target` metric from Application Load Balancer, a load balancing option for the Elastic Load Balancing service. Amazon EC2 Auto Scaling will then automatically adjust the number of Amazon EC2 instances as needed to maintain your target.

Scheduled Scaling

Scaling based on a schedule allows you to scale your application ahead of known load changes, such as the start of business hours, thus ensuring that resources are available when users arrive, or in typical development or test environments that run only during defined business hours or periods of time.

You can use APIs to scale the size of resources within an environment (vertical scaling). For example, you could scale up a production system by changing the instance size or class. This can be achieved by stopping and starting the instance and selecting the different instance size or class. You can also apply this technique to other resources, such as EBS volumes, which can be modified to increase size, adjust performance (IOPS), or change the volume type while in use.

Fleet Management

Fleet management refers to the functionality that automatically replaces unhealthy instances in your application, maintains your fleet at the desired capacity, and balances instances across Availability Zones. Amazon EC2 Auto Scaling fleet management ensures that your application is able to receive traffic and that the instances themselves are working properly. When AWS Auto Scaling detects a failed `health check`, it can replace the instance automatically.

Instances Purchasing Options

With Amazon EC2 Auto Scaling, you can provision and automatically scale instances across purchase options, Availability Zones, and instance families in a single application to optimize scale, performance, and cost. You can include Spot Instances with On-Demand and Reserved Instances in a single AWS Auto Scaling group to save up to 90 percent on compute. You have the option to define the desired split between On-Demand and Spot capacity, select which instance types work for your application, and specify preferences for how Amazon EC2 Auto Scaling should distribute the AWS Auto Scaling group capacity within each purchasing model.

Golden Images

A *golden image* is a snapshot of a particular state of a resource, such as an Amazon EC2 instance, Amazon EBS volumes, and an Amazon RDS DB instance. You can customize an Amazon EC2 instance and then save its configuration by creating an Amazon Machine Image (AMI). You can launch as many instances from the AMI as you need, and they will all include those customizations. A golden image results in faster start times and removes dependencies to configuration services or third-party repositories. This is important in auto-scaled environments in which you want to be able to launch additional resources in response to changes in demand quickly and reliably.

AWS Auto Scaling

AWS Auto Scaling monitors your applications and automatically adjusts capacity of all scalable resources to maintain steady, predictable performance at the lowest possible cost. Using AWS Auto Scaling, you can set up application scaling for multiple resources across multiple services in minutes.

AWS Auto Scaling automatically scales resources for other AWS services, including Amazon ECS, Amazon DynamoDB, Amazon Aurora, Amazon EC2 Spot Fleet requests, and Amazon EC2 Scaling groups.

If you have an application that uses one or more scalable resources and experiences variable load, use AWS Auto Scaling. A good example would be an ecommerce web application that receives variable traffic throughout the day. It follows a standard three-tier architecture with Elastic Load Balancing for distributing incoming traffic, Amazon EC2 for the compute layer, and Amazon DynamoDB for the data layer. In this case, AWS Auto Scaling scales one or more Amazon EC2 Auto Scaling groups and DynamoDB tables that are powering the application in response to the demand curve.

AWS Auto Scaling continually monitors your applications to make sure that they are operating at your desired performance levels. When demand spikes, AWS Auto Scaling automatically increases the capacity of constrained resources so that you maintain a high quality of service.

AWS Auto Scaling bases its scaling recommendations on the most popular scaling metrics and thresholds used for AWS Auto Scaling. It also recommends safe guardrails for scaling by providing recommendations for the minimum and maximum sizes of the resources. This way, you can get started quickly and then fine-tune your scaling strategy over time, allowing you to optimize performance, costs, or balance between them.

The *predictive scaling* feature uses machine learning algorithms to detect changes in daily and weekly patterns, automatically adjusting their forecasts. This removes the need for the manual adjustment of AWS Auto Scaling parameters as cyclicality changes over time, making AWS Auto Scaling simpler to configure, and provides more accurate capacity provisioning. Predictive scaling results in lower cost and more responsive applications.

DynamoDB Auto Scaling

DynamoDB automatic scaling uses the AWS Auto Scaling service to adjust provisioned throughput capacity dynamically on your behalf in response to actual traffic patterns. This enables a table or a global secondary index to increase its provisioned read and write capacity to handle sudden increases in traffic without throttling. When the workload decreases, AWS Auto Scaling decreases the throughput so that you don't pay for unused provisioned capacity.

Amazon Aurora Auto Scaling

Amazon Aurora automatic scaling dynamically adjusts the number of Aurora Replicas provisioned for an Aurora DB cluster. Aurora automatic scaling is available for both Aurora MySQL and Aurora PostgreSQL. Aurora automatic scaling enables your Aurora DB cluster to handle sudden increases in connectivity or workload. When the connectivity or workload decreases, Aurora automatic scaling removes unnecessary Aurora Replicas so that you don't pay for unused provisioned DB instances.

Amazon Aurora Serverless is an on-demand, automatic scaling configuration for the MySQL-compatible edition of Amazon Aurora. An Aurora Serverless DB cluster automatically starts up, shuts down, and scales capacity up or down based on your application's needs. Aurora Serverless provides a relatively simple, cost-effective option for infrequent, intermittent, or unpredictable workloads.

Accessing AWS Auto Scaling

There are several ways to get started with AWS Auto Scaling. You can set up AWS Auto Scaling through the AWS Management Console, with the AWS CLI, or with AWS SDKs.

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,