AWS Prescriptive Guidance

Provisioning production-ready Amazon EKS clusters using Terraform

AWS Prescriptive Guidance: Provisioning production-ready Amazon EKS clusters using Terraform

Table of Contents

Provisioning production-ready Amazon EKS clusters using Terraform

Jomcy Pappachen, Consultant, AWS Professional Services

Vara Bonthu, Senior Big Data Architect, AWS Professional Services

Ulaganathan N, Associate Consultant, AWS Professional Services

July 2021

Kubernetes is an open-source system for automating and managing containerized applications at scale. Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that runs container application workloads and helps standardize operations across your environments (for example, production or development environments). You can manage modern infrastructures by using infrastructure as code (IaC) practices with tools such as AWS CloudFormation, AWS Cloud Development Kit (CDK) , or Terraform by Hashicorp. This guide is intended for solution architects and technical leaders who are responsible for designing production-ready Amazon EKS clusters to run modernized workloads. The solution uses Terraform to build an IaC framework that provisions a multi-tenant Amazon EKS cluster. The guide describes the outcomes, design, architecture, and implementation of Amazon EKS clusters for running modernized application workloads.

By using this guide's solution, you can quickly create the infrastructure to migrate live-traffic serving self-hosted Kubernetes clusters to Amazon EKS on the AWS Cloud. The guide also provides a framework to help you design and create Amazon EKS clusters, each with a unique Terraform configuration and state file, in different environments across multiple AWS accounts and AWS Regions. When you want to modernize your applications with microservices and Kubernetes, you can use this guide and its reference code in the GitHub aws-eks-accelerator-for-terraform repository to build the Amazon EKS infrastructure on the AWS Cloud. This provisions Amazon EKS clusters, managed node groups with On-Demand and Spot Amazon Elastic Compute Cloud (Amazon EC2) instance types, AWS Fargate profiles, and plugins or add-ons for creating production-ready Amazon EKS clusters. The Terraform Helm provider also deploys common Kubernetes add-ons by using Helm charts.

The guide has the following four sections:

- Development environment requirements and code repository (p. 3) – Provides the software, tools, and GitHub repository to implement this guide's solution.
- High-level architecture (p. 5) – Explains the high-level architectural design of the guide's solution.
- Helm add-ons and Kubernetes Cluster Autoscaler (p. 7) – Describes how to implement the Helm modules by using Terraform Helm provider and how the Kubernetes Cluster Autoscaler helps scale Amazon EKS clusters.
- Logging and monitoring Amazon EKS clusters (p. 8) – Discusses the centralized logging and monitoring solutions that can be implemented for Amazon EKS clusters.

After provisioning the Amazon EKS clusters, you can use the code examples from the GitHub aws-eks-accelerator-for-terraform repository. However, this guide doesn't provide a complete overview of all implementations and we recommend that you carefully evaluate all third-party or open-source tools according to your organization's policies and requirements.

Solution outcomes

You should expect the following eight outcomes from deploying this guide’s solution in your AWS accounts:

- Enable your cross-functional teams to use the same Amazon EKS cluster by provisioning Amazon EKS clusters that support multi-tenancy based on applications and namespaces.
- Provision Amazon EKS clusters in new or existing virtual private clouds (VPCs), which means that you can use existing VPCs if required.
- Define your scaling metrics as a Kubernetes manifest by using Kubernetes Horizontal Pod Autoscaling and configurable options for expanding resource quotas and pod security policies.
- Ensure role-based access control (RBAC) for your developers and administrators by using AWS Identity and Access Management (IAM) roles.
- Deploy a private Amazon EKS cluster to secure your application and meet your compliance requirements.
- Monitor and log applications and system pods by using Amazon CloudWatch to collect and track metrics.
- Flexibly provision your Amazon EKS clusters with different node group types by running a combination of self-managed nodes, Amazon EKS managed node groups, and Fargate.
- Deploy a Bottlerocket Amazon Machine Image (AMI) in self-managed node groups to run container workloads in a purpose-built operating system (OS) on the AWS Cloud.

Development environment requirements and code repository

The following sections describe the software and tools required to set up your development environment, in addition to the tools required to validate and monitor your Amazon Elastic Kubernetes Service (Amazon EKS) clusters. An overview is also provided of the GitHub aws-eks-accelerator-for-terraform repository that contains the code for this guide's solution.

Development environment requirements

The following table shows the tools and versions to set up the development environment for building and deploying the guide's solution.

Tool	**Version**	**Purpose**
Git	2.31.1	Version control
Terraform	0.14.0	IaC
Helm 3	3.0.x	Kubernetes packaging
`kubectl`	1.18	Kubernetes command line interface
Kubernetes Lens	V4.2.2	User interface (UI) for cluster monitoring
IntelliJ IDEA Community Edition	2020.4	Integrated development environment (IDE)

Code repository for the solution

The code framework in the GitHub aws-eks-accelerator-for-terraform repository helps you to create Amazon EKS clusters, each with unique Terraform configuration and state files, in different environments across multiple AWS accounts and AWS Regions. The following list provides the outline of the repository's contents:

- The top-level live directory contains the configuration for each Amazon EKS cluster. Each folder under `live/<region>/application` represents an Amazon EKS cluster environment (for example, development or testing). This directory contains the `backend.conf` and `base.tfvars` files that create a unique Terraform state for each Amazon EKS cluster environment. You can update `backend.conf` with the Terraform backend configuration and `base.tfvars` with the Amazon EKS cluster common configuration variables.
- The `source` directory contains the `main.tf` main driver file.
- The `modules` directory contains the AWS resource modules.
- The `helm` directory contains the Helm chart modules.

- The `examples` directory contains sample template files with a `base.tfvars` file that you can use to deploy Amazon EKS clusters with multiple add-on options.
- The *How to deploy* section of the `ReadMe` file provides the step-by-step process and commands to provision the Amazon EKS clusters.

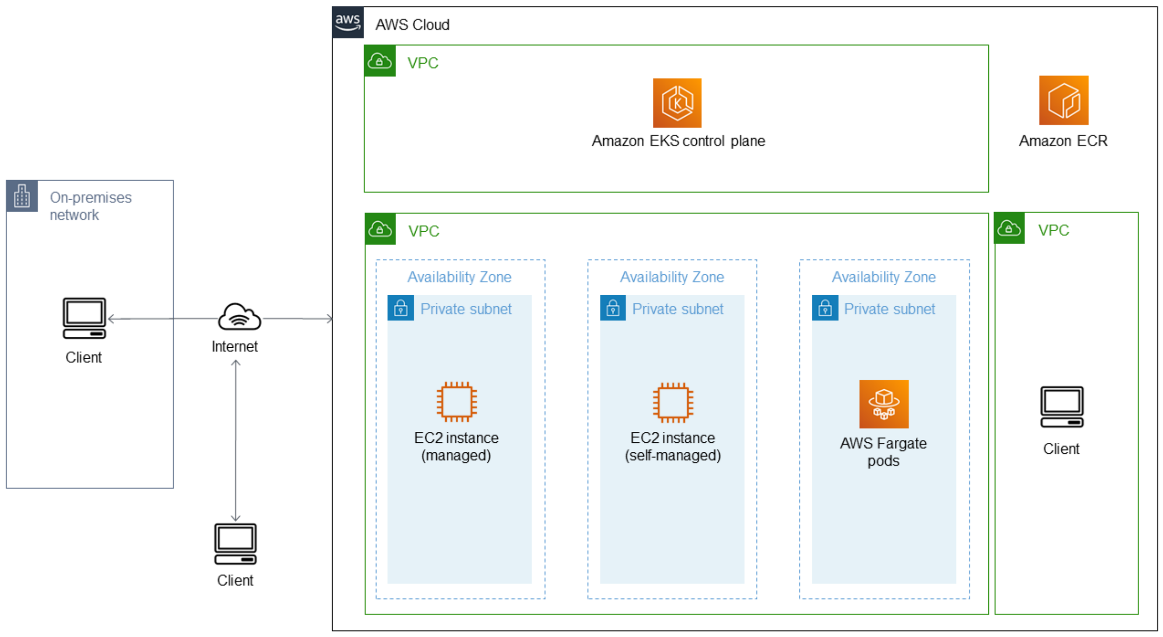
High-level architecture

The following sections explain the high-level architecture and components required for building the Amazon Elastic Kubernetes Service (Amazon EKS) clusters and Helm add-ons.

Typical Amazon EKS architecture

Typically, an Amazon EKS cluster consists of two main components, the *control plane* and the *data plane*, that run in their own individual virtual private clouds (VPCs). In Amazon EKS, the control plane is provided and maintained by AWS in a separate VPC. The nodes that you manage in your VPCs are responsible for running the container images or workloads. AWS also provides the required networking framework to integrate these components and create a Kubernetes cluster.

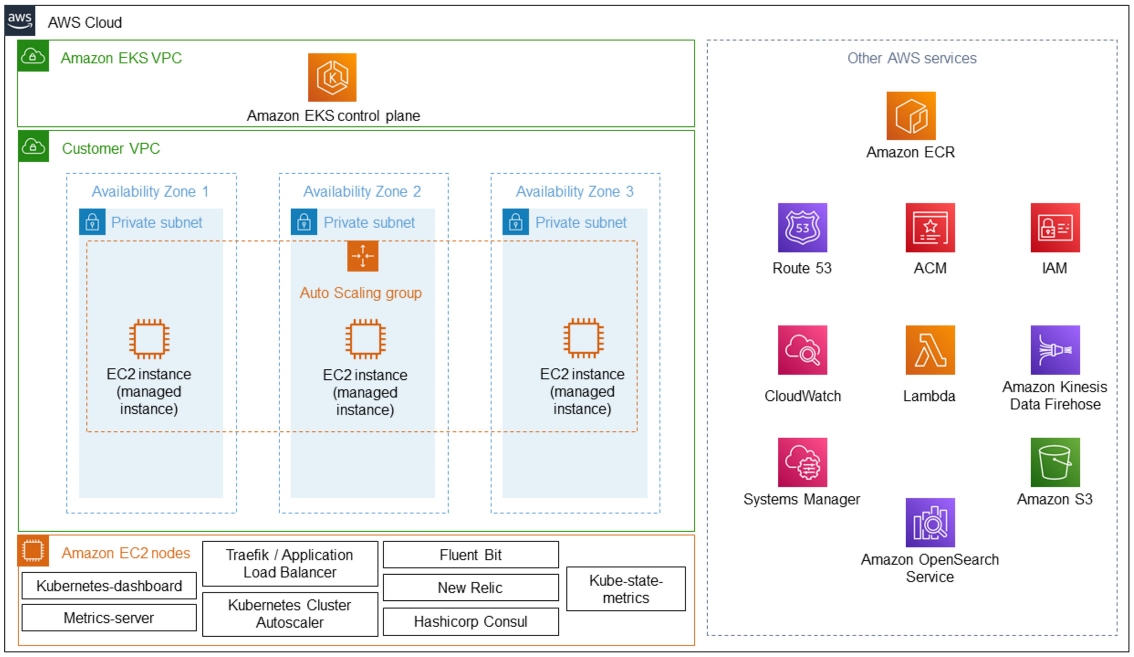
An Amazon EKS cluster can schedule pods on any combination of self-managed nodes, Amazon EKS managed node groups, and AWS Fargate. Amazon EKS nodes run in your account and connect to the cluster's control plane through the cluster’s API server endpoint. The following diagram shows the key components of an Amazon EKS cluster and the relationship between these components and a VPC.



For more information about Amazon EKS cluster networking, see Amazon EKS networking in the Amazon EKS documentation.

Planned high-level architecture

This section describes the high-level architecture for the guide’s solution, in addition to the AWS services and Helm modules that are used. The following diagram shows the high-level architecture for this solution.



The diagram shows the following components from this guide's solution:

- Amazon EKS clusters in different environments in AWS accounts across multiple AWS Regions, with a unique Terraform configuration and state file for each Amazon EKS cluster.
- One VPC with private subnets in each Availability Zone for nodes.
- VPC endpoints to access AWS services across AWS accounts.
- Managed node groups with On-Demand Instances.
- Managed node groups with Spot Instances.
- Fargate profiles run serverless workloads.
- Amazon Elastic Container Registry (Amazon ECR) stores the Docker images for application microservices and Helm add-ons for application deployments.
- On-Demand instances in an Amazon EC2 Auto Scaling group that are used as underlying computing infrastructure for the Amazon EKS cluster.
- Nodes deployed over multiple Availability Zones and using Amazon EC2 Auto Scaling groups.
- An Amazon Route 53 Domain Name System (DNS) zone for service discovery and a Network Load Balancer configured for HTTPS encrypted traffic.
- AWS Certificate Manager (ACM) to provision Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for secure communication.
- Kubernetes Metrics Server to collect metrics from running pods, such as CPU and memory utilization.
- Kubernetes Cluster Autoscaler to scale in and out of nodes.
- An Application Load Balancer ingress controller to load balance the application traffic.
- Amazon CloudWatch with Fluent Bit for logging application logs and cluster logs.
- Amazon OpenSearch Service (successor to Amazon Elasticsearch Service) and Amazon Simple Storage Service (Amazon S3) for centralized logging.

Helm add-ons and Kubernetes Cluster Autoscaler

Helm is a package manager for Kubernetes that helps you install and manage applications in your Kubernetes cluster. Helm packages multiple Kubernetes resources into a single logical deployment unit called a chart. This guide's solution helps you to launch an Amazon Elastic Kubernetes Service (Amazon EKS) cluster with the following Helm charts.

Chart name	Namespace	Chart version	Application version	Docker version
cluster-autoscaler	`kube-system`	9.9.2	1.19.1	v1.19.1
aws-for-fluent-bit	`logging`	0.1.7	2.6.1	2.12.0
metric-server	`kube-system`	2.11.4	0.3.6	v0.3.6
newrelic-infrastructure	`kube-system`	1.3.1	1.26.2	1.26.2
AWS Load Balancer Controller	`aws-load-balancer-controller`	1.1.6	v2.1.3	v2.1.3

Important
This guide's tool or component versions can change and might switch to other tools with similar capabilities.

Kubernetes Cluster Autoscaler

The Kubernetes Cluster Autoscaler is an add-on that adjusts the size of a Kubernetes cluster to meet your workload resource requirements. Kubernetes Cluster Autoscaler increases the size of the cluster when pods failed to schedule on current nodes due to insufficient resources and it also attempts to remove underutilized nodes.

Kubernetes Cluster Autoscaler automatically scales Amazon Elastic Compute Cloud (Amazon EC2) instances according to the resource requirements of the pods. You can use Kubernetes Cluster Autoscaler to control scaling activities by changing the required capacity of the Amazon EC2 Auto Scaling group and directly terminating instances. Each node group maps to a single Amazon EC2 Auto Scaling group; however, Kubernetes Cluster Autoscaler requires that all EC2 instances in a node group share the same vCPU number and RAM amount.

Logging and monitoring Amazon EKS clusters

Logging and monitoring for Amazon Elastic Kubernetes Service (Amazon EKS) has two categories: the *control plane logs* and the *application logs*. Amazon EKS control plane logging provides audit and diagnostic logs from the control plane to Amazon CloudWatch Logs groups in your AWS account. To collect application logs you must install a log aggregator, such as Fluent Bit, Fluentd, or CloudWatch Container Insights, in your Amazon EKS cluster.

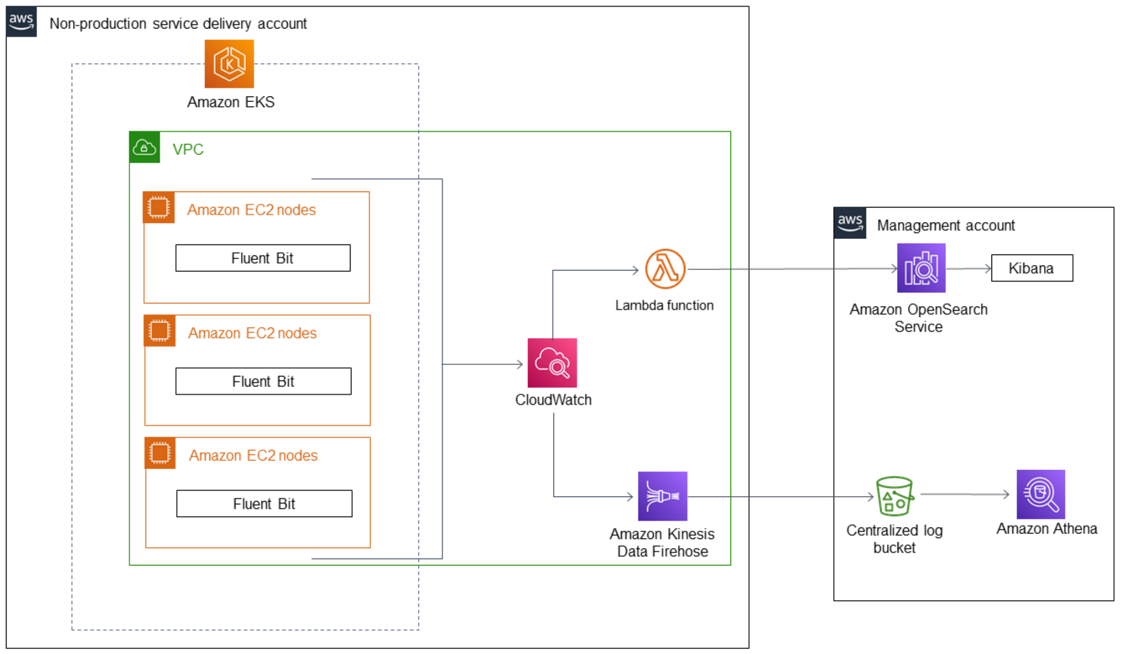
Fluent Bit is an open-source log processor and forwarder that is written in C++, which means that you can collect data from different sources, enrich them with filters, and send them to multiple destinations. By using this guide's solution you can enable `aws-for-fluent-bit` or `fargate-fluentbit` for logging. Fluentd is an open-source data collector for unified logging layer and written in Ruby. Fluentd acts as a unified logging layer that can aggregate data from multiple sources, unify data with different formats into JSON-formatted objects, and route them to different output destinations.

Choosing a log collector is important for CPU and memory utilization when you monitor thousands of servers. If you have multiple Amazon EKS clusters, you can use Fluent Bit as a lightweight shipper to collect data from different nodes in the cluster and forward it to Fluentd for aggregation, processing and routing to a supported output destination.

It's very important to monitor and maintain the performance and health of your scalable Kubernetes environments. You can monitor the Amazon EKS clusters in real time by streaming metrics to New Relic or Datadog for better observability. New Relic and Datadog provide holistic views of the performance and health of Kubernetes clusters, down to the node, container, and application-level visibility required to identify and troubleshoot performance issues. We recommend using Fluent Bit as a log collector and using New Relic or Datadog for better observability. The following list provides three options for logging and monitoring your Amazon EKS clusters:

- **Option 1** – Use Fluent Bit as the log collector and forwarder to send application and cluster logs to CloudWatch. You can then stream the logs to Amazon OpenSearch Service (successor to Amazon Elasticsearch Service) using a subscription filter in CloudWatch. This option is shown in this section's architecture diagram.
- **Option 2** – Use a Datadog agent as the log and metric collector and forwarder to stream logs and metrics to the Datadog UI.
- **Option 3** – Use a New Relic agent as the log and metric collector and forwarder to stream logs and metrics to the New Relic UI.

The following diagram shows a logging architecture that automatically streams logs from multiple accounts to a centralized logs server.



The diagram shows the following workflow when application logs from Amazon EKS clusters are streamed to Amazon OpenSearch Service:

1. The Fluent Bit service in the Amazon EKS cluster pushes the logs to CloudWatch.
2. The AWS Lambda function streams the logs to Amazon OpenSearch Service using a subscription filter.
3. You can then use Kibana to visualize the logs in the configured indexes.
4. You can also stream logs by using Amazon Kinesis Data Firehose and store them in an S3 bucket for analysis and querying with Amazon Athena.

Resources

- Logging for Amazon Elastic Kubernetes Service (Amazon EKS)
- Centralized container logging with Fluent Bit
- Terminate HTTPS traffic on Amazon EKS workloads with AWS Certificate Manager (ACM)
- Set up end-to-end TLS encryption on Amazon EKS with the AWS Load Balancer Controller
- Amazon EKS best practices guide for security
- Amazon EKS platform versions
- Designing and implementing logging and monitoring with Amazon CloudWatch

AWS Prescriptive Guidance glossary

AI and ML terms (p. 11) | Migration terms (p. 12) | Modernization terms (p. 16)

AI and ML terms

The following are commonly used terms in artificial intelligence (AI) and machine learning (ML)-related strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

binary classification	A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"
classification	A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.
data preprocessing	To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.
deep ensemble	To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.
deep learning	An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.
exploratory data analysis (EDA)	The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.
features	The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.
feature importance	How significant a feature is for a model’s predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see Machine learning model interpretability with AWS.

feature transformation	To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the “2021-05-27 00:15:37” date into “2021”, “May”, “Thu”, and “15”, you can help the learning algorithm learn nuanced patterns associated with different data components.
interpretability	A characteristic of a machine learning model that describes the degree to which a human can understand how the model’s predictions depend on its inputs. For more information, see Machine learning model interpretability with AWS.
multiclass classification	A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"
regression	An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).
training	To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don’t know the target.
target variable	The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.
tuning	To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.
uncertainty	A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the Quantifying uncertainty in deep learning systems guide.

Migration terms

The following are commonly used terms in migration-related strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

7 Rs	Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following: • Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.

	• Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud. • Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com. • Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud. • Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. This migration scenario is specific to VMware Cloud on AWS, which supports virtual machine (VM) compatibility and workload portability between your on-premises environment and AWS. You can use the VMware Cloud Foundation technologies from your on-premises data centers when you migrate your infrastructure to VMware Cloud on AWS. Example: Relocate the hypervisor hosting your Oracle database to VMware Cloud on AWS. • Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them. • Retire – Decommission or remove applications that are no longer needed in your source environment.
application portfolio	A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.
artificial intelligence operations (AIOps)	The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.
AWS Cloud Adoption Framework (AWS CAF)	A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.
AWS landing zone	A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.
AWS Workload Qualification Framework (AWS WQF)	A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema

	Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.
business continuity planning (BCP)	A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.
Cloud Center of Excellence (CCoE)	A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.
cloud stages of adoption	The four phases that organizations typically go through when they migrate to the AWS Cloud: • Project – Running a few cloud-related projects for proof of concept and learning purposes • Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model) • Migration – Migrating individual applications • Re-invention – Optimizing products and services, and innovating in the cloud These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.
configuration management database (CMDB)	A database that contains information about a company's hardware and software products, configurations, and inter-dependencies. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.
epic	In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.
heterogeneous database migration	Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.
homogeneous database migration	Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.
idle application	An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.
IT information library (ITIL)	A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)	Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.
large migration	A migration of 300 or more servers.
Migration Acceleration Program (MAP)	An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.
Migration Portfolio Assessment (MPA)	An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.
Migration Readiness Assessment (MRA)	The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.
migration at scale	The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.
migration factory	Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the CloudEndure Migration Factory guide in this content set.
migration metadata	The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.
migration pattern	A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.
migration strategy	The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs (p. 12) entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.
operational-level agreement (OLA)	An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).
operations integration (OI)	The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.

organizational change management (OCM)	A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.
playbook	A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.
portfolio assessment	A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see Evaluating migration readiness.
responsible, accountable, consulted, informed (RACI) matrix	A matrix that defines and assigns roles and responsibilities in a project. For example, you can create a RACI to define security control ownership or to identify roles and responsibilities for specific tasks in a migration project.
runbook	A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.
service-level agreement (SLA)	An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.
task list	A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.
workstream	Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.
zombie application	An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.

Modernization terms

The following are commonly used terms in modernization-related strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

business capability	What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.
domain-driven design	An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley

	Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.
microservice	A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.
microservices architecture	An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.
modernization	Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.
modernization readiness assessment	An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.
monolithic applications (monoliths)	Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.
polyglot persistence	Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.
split-and-seed model	A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.
strangler fig pattern	An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an

	example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.
two-pizza team	A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development. For more information, see the Two-pizza team section of the Introduction to DevOps on AWS whitepaper.

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

update-history-change	update-history-description	update-history-date
— (p. 19)	Initial publication	July 23, 2021