AWS Whitepaper

Containers on AWS

Containers on AWS: AWS Whitepaper

Table of Contents

Containers on AWS

Publication date: **March 8, 2024** (*Document history*)

Abstract

This whitepaper provides guidance and options for running containers on AWS. Containers provide a way to develop, ship, and run applications in an isolated environment. AWS is a natural complement to containers and offers a wide range of scalable orchestration and infrastructure services, upon which containers can be deployed. This paper provides information about container orchestration and compute options such as AWS App Runner, Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and AWS Fargate and key considerations for container workloads on AWS.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

Introduction

Before containers were an option, developers and administrators faced challenges with compatibility restrictions. You would build applications specifically for their pre-determined environments. When a workload needed to be migrated (for example, from bare metal to a virtual machine (VM), from a VM to the cloud, or between service providers), you had to rebuild the application or the workload entirely to ensure compatibility with the new environment. Container technology was invented to overcome these incompatibilities by providing a common interface. With the release of Docker, the interest in containers technology rapidly increased.

Container technology uses the resource-isolation features of the Linux kernel to sandbox an application, its dependencies, configuration files, and interfaces inside an atomic unit called a container. This allows a container to run on any host with the suitable kernel components, while

shielding the application from behavioral inconsistencies through variances in software installed on the host. Containers use operating system (OS) level virtualization compared to VMs, which use hardware level virtualization using hypervisor. A hypervisor is a software or a firmware that creates and runs VMs. Multiple containers can run on a single host OS without needing a hypervisor, while isolated from neighboring containers. This layer of isolation allows consistency, flexibility, and portability, which enable rapid software deployment and testing. There are many ways in which using containers on AWS can benefit your organization. Containers have been widely employed in use cases such as distributed applications, batch jobs, and continuous deployment pipelines. The use cases for containers continue to grow in areas like distributed data processing, streaming media delivery, genomics, and machine learning, including generative AI.

Container benefits

The rapid growth of containers is fueled by the many benefits it provides. If you have applications that run on VMs or bare metal servers today, you can consider containerizing them to take advantages of the benefits from containers. The primary benefits of container are speed, consistency, density, resource efficiency, and portability.

Speed

Thanks to its lightweight and modular nature, containers can enable rapid iteration of your applications. Your development speed improves through the ability to deconstruct applications into smaller units. This is advantageous because it reduces shared resources among application components, leading to fewer compatibility issues between required libraries or packages. The container startup time primarily depends on the size of the container image, cache, the time to pull the image and start the container on host. To improve the container startup time, keep the size of your container image as small as possible. You can use techniques like multi-stage builds, local cache, and Seekable OCI (SOCI) when applicable.

Consistency

The consistency and fidelity of a modular development environment provides predictable results when moving code between development, test, and production systems. By ensuring that the container encapsulates exact versions of necessary libraries and packages, it is possible to minimize the risk of any compatibility issue.

This concept easily lends itself to a *disposable system* approach, in which patching individual containers is less preferable than building new containers in parallel, testing, and replacing the old. This practice helps avoid *drift* of packages across a fleet of containers, versions of your application, or dev/test/prod environments; the result is more consistent, predictable, and stable applications.

Density and resource efficiency

Containers facilitate enhanced resource efficiency by allowing multiple containers to run on a single host system. Resource efficiency is a natural result of the isolation and allocation techniques that containers use. You can restrict your containers to a required number of CPUs and allocate specific amount of memory. By understanding the resources needs for your container and the

resources available to your VM or the underlying host, it's possible to maximize the containers running on a single host. This results in higher density, increased efficiency of compute resources, and less wastage on excess capacity. We will discuss more about container scheduling details in the Scheduling section.

Portability

The flexibility of containers is based on its portability, ease of deployment, and smaller size compared to virtual machines. The Open Container Initiative (OCI), was formed to support fully interoperable container open standards with 3 specifications:

- The Runtime Specification (runtime-spec)
- The Image Specification (image-spec)
- The Distribution Specification (distribution-spec)

The OCI image specification defines an OCI Image, consisting of an image manifest, which contains metadata about contents and dependencies of the image; an image index (optional); a set of filesystem layers; and a configuration, such as arguments and environment variables. You can run the OCI compliant container image on any supported version of Linux or Windows, if you have the OCI compliant container runtime installed on the host.

OCI allows a compliant container to be portable across all major operating systems and platforms without worrying that their current choice of any infrastructure, cloud provider, or tool will lock them into any technology provider as a container format is not bound to any higher level constructs.

Containers also provide the flexibility which makes micro-services architecture possible. In common infrastructure models, a virtual machine runs multiple services. In contrast, containers package services in isolation, on top of a common host OS. This allows a service to move between hosts, stay isolated from failure of other adjacent services, and stay protected from errant patches or software upgrades on the host system. Because containers provide clean, reproducible, and modular environments, it streamlines both code deployment and infrastructure management.

Containers services on AWS

AWS is an elastic, secure, flexible, and developer-centric cloud provider, which makes it ideal for container workloads. AWS offers scalable infrastructure, APIs, and SDKs that integrate into the development lifecycle and accentuate the benefits that containers offer. In this section, we will discuss the different options for container deployments using AWS services:

- **AWS App Runner** is a fully managed service that makes it easy to quickly deploy containerized web applications and APIs. You can use an existing container image, container registry, source code repository, or existing CI/CD workflow to quickly run a fully containerized web application. App Runner supports full stack development, with both front-end and back-end web applications that use HTTP and HTTPS protocols. App Runner automatically builds and deploys the web application and load balances traffic with encryption. It monitors the number of concurrent requests sent to your application and automatically adds additional instances based on request volume. When your application receives no incoming requests, App Runner scales the containers down to a CPU throttled instance, which can serve incoming requests within milliseconds. App Runner is ideal when you want to run and scale your application on AWS without configuring or managing infrastructure services. You do not have to configure any orchestrators, set up build pipelines, manage load balancers, or rotate TLS certificates. When you associate a source code repository with App Runner, it can automatically containerize your web application and run it. This makes it the simplest way to build and run your containerized web application on AWS.
- **Amazon Elastic Container Service** (Amazon ECS) is a fully managed container orchestration service which provides a convenient way to rapidly launch thousands of containers across a broad range of AWS compute options. You can use your preferred CI/CD and automation tools with Amazon ECS. There is no complexity of managing a control plane, add-ons, and nodes with Amazon ECS. Amazon ECS offers two launch types – Amazon EC2 and AWS Fargate (discussed later). With the Amazon EC2 launch type, Amazon ECS provides an easy lift for your applications that run on VMs. Your Amazon ECS clusters have container instances, which are Amazon EC2 instances running an Amazon ECS container agent. The agent communicates instance and container state information to the cluster manager. The Amazon ECS container agent is included in the Amazon ECS-optimized AMI, but you can also install it on any Amazon EC2 instance that supports the Amazon ECS specification. Your containers are defined in an Amazon ECS task definition that you use to run individual tasks or tasks within an Amazon ECS service. An Amazon ECS service enables you to run and maintain a specified number of tasks simultaneously in a cluster. The task definition can be thought of as a blueprint for your application that you can

specify parameters such as the container image to use, which ports to open, the amount of CPU and memory to use with each task or containers within a task, and the IAM role the task should use.

Amazon ECS also supports hybrid deployment scenarios. You can manage your containers on-premises using Amazon ECS Anywhere. Additionally, you also have options to deploy containers on Outposts, Local Zones, and Wavelength with Amazon ECS. Since we are focused on container deployments in the cloud, the details of these hybrid options is beyond the scope of this whitepaper. Refer to links in the ??? section for more information on this topic.

- **Amazon Elastic Kubernetes Service** (Amazon EKS) is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes. It provides a natural migration path if you use Kubernetes already and want to continue using it on AWS for your container applications. It provides highly-available and secure clusters and automates key tasks such as security patching, node provisioning, and updates. Amazon EKS runs a single-tenant Kubernetes control plane for each cluster. The control plane infrastructure is not shared across clusters or AWS accounts. Amazon EKS automatically manages the availability and scalability of the Kubernetes control plane nodes, which are responsible for scheduling containers, managing the availability of applications, storing cluster data, and other key operations. You can also use the managed node group option in Amazon EKS to automate the provisioning and lifecycle management of worker nodes that run your pods. Amazon EKS runs upstream Kubernetes, certified conformant for a predictable experience. You can easily migrate any standard Kubernetes application to Amazon EKS without refactoring your code. This allows you to deploy and manage workloads on your Amazon EKS cluster the same way that you would with any other Kubernetes environment. Currently supported versions are listed in the Amazon EKS user guide. To support operational capabilities on Kubernetes clusters, customers can leverage Amazon EKS add-ons, a curated set of software that simplifies management of operational activities on clusters. For details on how to use add-ons, please refer to EKS add-ons.

  Amazon EKS also supports hybrid deployment scenarios. You can manage your containers on-premises using Amazon EKS Anywhere. Amazon EKS Anywhere makes use of Amazon EKS Distro, which is the open source distribution of Kubernetes built and maintained by AWS. This is the same distribution which is used in Amazon EKS on AWS. Amazon EKS Anywhere provides an installable software package for creating and operating Kubernetes clusters on-premises and automation tooling for cluster lifecycle support. Additionally, you have options to deploy Amazon EKS on Outposts, Local Zones, and AWS Wavelength. These hybrid options are beyond the scope of this whitepaper. Refer to links in the ??? section for more information on this topic.

- **AWS Fargate** provides a fully managed compute option to run containers for both Amazon ECS and Amazon EKS. Fargate reduces the time spent on configuration, patching, and security. Fargate runs each task or pod in its own kernel, providing the tasks and pods their own isolated compute environment. This enables your application to have workload isolation and improved security by design. With Fargate, there is no over-provisioning and paying for additional servers. It allocates the right amount of compute, eliminating the need to choose instances and scale cluster capacity. With the Fargate launch type, you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task. For Amazon EKS, Fargate integrates with Kubernetes using controllers that are built by AWS using the extension model provided by Kubernetes. These controllers run as part of the Amazon EKS managed Kubernetes control plane and are responsible for scheduling native Kubernetes pods onto Fargate compute.

Other options: There are additional container deployment offerings available on AWS, which can be useful based on the nature of your workloads. AWS offers extensive documentation and blog posts available for each of these offerings.

- **AWS Batch** helps you to run batch computing workloads on the AWS Cloud. You can define job definitions that specify which container images to run your jobs, which run as containerized applications on AWS Fargate or Amazon EC2 resources in your compute environment.
- **AWS Elastic Beanstalk** supports the deployment of web applications from containers. With containers, you can define your own runtime environment. You can also choose your own platform, programming language, and any application dependencies (such as package managers or tools), which typically aren't supported by other platforms.
- **AWS Lambda** functions can be packaged and deployed as container images of up to 10 GB in size. This allows you to easily build and deploy larger workloads that rely on sizable dependencies, such as machine learning or data intensive workloads. Just like functions packaged as ZIP archives, functions deployed as container images benefit from the same operational simplicity, automatic scaling, high availability, and native integrations with many services that you get with Lambda.
- **Amazon Lightsail** is a highly scalable compute and networking resource on which you can deploy, run, and manage containers. When you deploy your images to your Lightsail container service, the service automatically launches and runs your containers in the AWS infrastructure.

- **Red Hat OpenShift Service on AWS (ROSA)** can accelerate your application development process if you are presently running containers in OpenShift by leveraging familiar OpenShift APIs and tools for deployments on AWS. ROSA comes with pay-as-you-go hourly and annual billing, a 99.95% SLA, and joint support from AWS and Red Hat.

Your choice of service is driven by your workload properties, the ease of getting started and the amount of control and customization flexibility you want to have. Consider starting on the fully-managed end of the spectrum (App Runner orFargate) and work backwards towards a more self-managed experience based on the demands of your workload. The self-managed experience can go to the extent of even running containers directly on virtual machines with Amazon EC2, without using any AWS managed services. With AWS, you have the flexibility to pick the container deployment that works best for your operational needs without compromising on the benefits of containers.

Key considerations

Container runtime

A container runtime, also known as container engine, is a software component that can run containers on a host operating system. Container runtimes are responsible for loading container images from a repository, monitoring local system resources, isolating system resources for use of a container, and managing container lifecycle. They come in two forms:

- High-level container runtimes (such as *containerd* and *CRI-O*) provide functions that run on top of low-level runtime.
- Low-level runtimes are responsible for creating and running containers. The primary job of the low-level container runtimes is to provide container lifecycle management. These runtimes implement the Runtime Specification provided by the OCI(Open Container Initiative), a Linux Foundation project started by Docker, which aims to provide open standards for Linux containers. The default reference implementation for low level runtimes specified by OCI is *runc*.

It's important to note that Kubernetes and Amazon EKS have started using *containerd* as the default runtime from version 1.24. This shouldn't impact your existing images if they are OCI compliant. Refer to the blog post All you need to know about moving to containerd on Amazon EKS for key factors when migrating to *containerd*.

Container-enabled AMIs

AWS has developed a streamlined, purpose-built operating system for use with Amazon Elastic Container Service. The Amazon ECS-optimized AMI is built on top of Amazon Linux 2 and Amazon Linux 2023. It is pre-configured with the Amazon ECS container agent and containerd daemon with runtime dependencies. The Amazon EKS-optimized Amazon Linux AMI is based on Amazon Linux 2. It includes kubelet, AWS IAM Authenticator, Docker (for Amazon EKS version 1.23 and earlier), and containerd.

Although you can create your own custom AMIs that meet the basic specifications needed to run your containerized workloads, the Amazon ECS and Amazon EKS-optimized AMIs are pre-configured with requirements and recommendations tested by AWS engineers. You can also use Bottlerocket, a Linux-based open-source operating system purpose-built by AWS for running

containers. Bottlerocket includes only the essential software required to run containers and ensures that the underlying software is always secure. With Bottlerocket, you can reduce maintenance overhead and automate workflows by applying configuration settings consistently as nodes are upgraded or replaced.

You can run Windows containers on both Amazon EKS and Amazon ECS. This helps anyone who wants to run Windows workloads on Amazon EKS and Amazon ECS. Amazon ECS supports Windows containers on Fargate, but they are not supported for Amazon EKS Pods on Fargate. Refer to the best practices guide and key considerations before running Windows containers on Amazon EKS.

Compute options

From a compute choice perspective, AWS offers a wide range of instance types for running applications. AWS also offers purpose-built compute for meeting specific requirements. For instance, customers can leverage Nitro-based Amazon EC2 instances for running IPv6-enabled Amazon EKS clusters.

Graviton

Graviton-powered instances provide the best price performance across a wide variety of general-purpose, compute-optimized, memory-optimized and accelerated compute instances. Graviton2 processors on Amazon EC2 general purpose (M6g), compute-optimized (C6g), and memory-optimized (R6g) Amazon EC2 instances deliver up to 40% improved price/performance over current generation M5, C5, and R5 instances. Customers can leverage Graviton on AWS-managed services (Amazon RDS, Aurora, MemoryDB) and AWS compute options (Amazon EKS, Amazon ECS, Amazon EMR, Lambda, Fargate, Elastic Beanstalk). Amazon EC2 instances powered by next-generation ARM64-based Graviton processors are well-suited for a broad spectrum of workloads, including application servers, open source databases, in-memory caches, microservices, gaming servers, electronic design automation, high-performance computing, and video encoding. Both Graviton instance generations are available to Amazon EKS. Running performance-optimized workloads on the latest Graviton instances, you can observe up to 19% performance increase and up to 15% cost savings compared to Graviton2 instances. Running containers on Graviton 3 instances also helps build a sustainable architecture that's 60% more energy efficient over comparable Amazon EC2 instances.

AWS Graviton has broad support for using its instance types on Amazon ECS and Amazon EKS. Amazon ECR also supports storing multi-architecture images including images built for arm

architectures and can use Graviton compute. Graviton is also supported on container-optimized operating systems like BottleRocket and on serverless AWS compute services like Fargate.

Compute for specialized workloads

With the increasing importance of use cases in generative AI, machine mearning (ML) and big data and analytics to global businesses and industries, containers have emerged as a popular option to run data workloads. These workloads are often high-throughput, compute-intensive, and critical to business operations, requiring the right configuration to support their requirements. AWS container services such as Amazon ECS and Amazon EKS have native integration with AWS Batch to support batch-oriented workloads. For workloads that require GPUs, both Amazon ECS and Amazon EKS support using GPU-based instances. In addition, you can leverage Amazon EKS and Amazon ECS with instance types such as Inferentia and Trainium to run machine-learning workloads at scale, with comparable performance at a much lower cost to GPU instances.

You can also take advantage of projects and frameworks such as Data on EKS (DoEKS) to simplify and speed up the process of building, deploying, and scaling data workloads on Amazon EKS. DoEKS offers IaC templates in Terraform and AWS Cloud Development Kit (AWS CDK), performance benchmark reports, best practices, and sample code to help users run applications like Spark, Kubeflow, MLFlow, Airflow, Presto, Kafka, Cassandra, and more on Amazon EKS with ease.

Scheduling

When applications need to scale out across multiple hosts, you need to be able to easily manage each additional container and node while abstracting the complexity of the underlying infrastructure. In this environment, *scheduling* refers to the ability to schedule containers on the most appropriate host in a scalable, automated way. In this section, we will review key scheduling aspects of various AWS container services.

- **Amazon ECS** provides flexible scheduling capabilities by leveraging the same cluster state information provided by the Amazon ECS APIs to make appropriate placement decision. Amazon ECS provides two scheduler options: service scheduler and the Run Task.
  - Service scheduler is suited for long-running stateless applications, which ensures the required number of tasks are always running (replica) and automatically reschedules if tasks fail. Services also let you deploy updates, such as changing the number of running tasks or the task definition version that should be running. The daemon scheduling strategy deploys exactly one task on each active container instance.

- The Run Task option is suited for batch jobs, scheduled jobs or a single job that perform work and stop. You can allow the default task placement strategy to distribute tasks randomly across your cluster, which minimizes the chances that a single instance gets a disproportionate number of tasks. Alternately, you can customize how the scheduler places tasks using task placement strategies and constraints. This enables you to optimize placement of containers to be as cost-efficient as possible by ensuring that your tasks are running on the instance types most suitable for your workload.

  The binpack placement strategy, for instance, tries to optimize placement of containers to be cost-efficient as possible. Containers in Amazon ECS are part of Amazon ECS tasks, placed on compute instances to leave the least amount of unused CPU or memory. This in turn minimizes the number of computed instances in use, resulting in better resource efficiency. The placement strategies can be supported by placement constraints, which lets you place tasks by constraints like the instance type or the availability zone.

- **Amazon EKS**: With Amazon EKS, the Kubernetes scheduler (*kube-scheduler*) is responsible for finding the best node for every newly created pod or any unscheduled pods that have no node assigned. It assigns the pod to the node with the highest ranking based on the filtering and ranking system. If there is more than one node with equal scores, *kube-scheduler* selects one of these at random. You can constrain a pod so that it can only run-on a set of nodes. The scheduler will automatically find a reasonable placement, but there are some circumstances where you might want to control which node the pod deploys to. For example, to ensure that a pod ends up on a machine with SSD storage attached to it, or to co-locate pods from two different services that communicate a lot into the same availability zone. Here are some important concepts about scheduling in Kubernetes:

  - **NodeSelector** is the simplest recommended form of node selection constraint. For the pod to be eligible to run on a node, the node must have each of the indicated key-value pairs as labels.
  - **Topology spread constraints** are to control how pods are spread across your cluster among failure-domains such as Regions, zones, nodes, and other user-defined topology domains. This can help to achieve high availability as well as efficient resource utilization.
  - **Node affinity** is a property of pods that attracts them to a set of nodes (either as a preference or a hard requirement). Taints are the opposite; they allow a node to repel a set of pods. Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

- **Pod Priority** indicates the importance of a pod relative to other pods. If a pod can't be scheduled, try to preempt or evict lower priority pods to make scheduling of the pending pod possible.

- With **Fargate**, **App Runner**, and **Lambda**, you don't need to manage how to schedule your containers. Compute is provisioned for you automatically as required, based on the resource requirements you have configured. The containers are automatically scheduled on provisioned compute.

Container repositories

Containers are distributed using container images. Images are a compile time construct, defined by the Dockerfile manifest. It is made up of a set of instructions to create the containers. Images are stored in container registries. Within a registry, a collection of related images is grouped together as repositories. When an image is run as a container, it is pulled from the container registry and stored locally in the compute.

Amazon Elastic Container Registry (Amazon ECR) is the AWS native managed container registry for Open Container Initiative (OCI) artifacts. It not only allows you to store container images, but also OCI artifacts such as Helm charts (a collection of files that describe a related set of Kubernetes resources) and open policy agent (OPA) bundles (tarballs that contain policies and data). With Amazon ECR, you can share container images privately within your organization using a private repository. By default, it is only accessible within your AWS account by IAM users with the necessary permissions. Amazon ECR also offers a public registry called Amazon ECR Public Gallery. Public repositories are available worldwide for anyone to discover and download. Amazon ECR comes with features like encryption at rest using AWS Key Management Service (AWS KMS) and in-transit using Transport Layer Security (TLS) endpoints. Using image lifecycle policies, you can automatically preserve the most recent images and archives the ones you don’t need. You can use rules and tagging to access container images faster.

Amazon ECR image scanning helps in identifying software vulnerabilities in your container images by using CVEs database from the Clair project and provides a list of scan findings. Additionally, you can use VPC interface endpoints for Amazon ECR to restrict the network traffic between your VPC and Amazon ECR to Amazon network, without a need for an internet gateway, NAT gateway, or a VPN or Direct Connect. You can also use a registry of your choice such as DockerHub or any other cloud of self-hosted container registry and integrate seamlessly with AWS container services.

Other notable features of Amazon ECR include support for pull through cache, image replication and support for multi-architecture container images. With Amazon ECR's pull-through cache repositories, you can retrieve, store, and sync container artifacts stored in publicly accessible container registries. With frequent registry syncs and no additional tools to manage, pull-through cache repositories help you keep container images sourced from public registries up to date. You can configure your Amazon ECR private registry to support the replication of your Amazon ECR repositories. Amazon ECR supports both cross-Region and cross-account replication. Amazon ECR also supports creating and pushing Docker manifest lists, which are used for multi-architecture images. A manifest list is a list of images that is created by specifying one or more image names. In most cases, the manifest list is created from images that serve the same function, but for different operating systems or architectures.

Observability

Observability is to continuously discover actionable insights based on signals from the system under observation over time. The signals can be low-level such as CPU, memory, disk space, and higher-level such as business signals including API response times, error rates, and transactions per second. Having insight into metrics, logs, and traces, understanding what is happening not just at the cluster or host level, but also within the container runtime and application, helps organizations make better informed decisions, such as when to scale in/out nodes/tasks/pods, change Amazon EC2 instance types, and purchasing options (on-demand, reserved, and spot).

Treating logs as a continuous stream of events instead of as static files allows you to react to the continuous nature of log generation. You can capture, store, and analyze real-time log data to get meaningful insights into the application's performance, network, and other characteristics. An application must not be required to manage its own log files.

You can specify the `awslogs` log driver for containers in your Amazon ECS task definition under the **logConfiguration** object to ship the `stdout` and `stderr` I/O streams to a designated log group in Amazon CloudWatch logs for viewing and archival. Additionally, FireLens for Amazon ECS enables you to use task definition parameters with the `awsfirelens` log driver to route logs to other AWS services or third-party log aggregation tools for log storage and analytics. FireLens works with **Fluentd** and **Fluent Bit**, fully compatible with Kubernetes. Using the **Fluent Bit** daemonset, you can send container logs from your Amazon EKS clusters to CloudWatch logs.

Amazon CloudWatch is a monitoring service that you can use to collect various system, and application-wide metrics and logs, and set alarms. CloudWatch Container Insights helps you explore, aggregate, and summarize your container metrics, application logs and performance log

events at the cluster, node, pod, task, and service level through automated dashboards in the CloudWatch console. Container Insights also provides diagnostic information, such as container restart failures, crashloop backoffs in an Amazon EKS cluster to help you isolate issues and resolve them quickly. Container Insights is available for Amazon Elastic Container Service (Amazon ECS, including Fargate), Amazon Elastic Kubernetes Service (Amazon EKS), and Kubernetes platforms on Amazon EC2.

AWS X-Ray provides a complete view of requests as they flow through your distributed applications with end-to-end tracing capabilities. You can filter visual data across payloads, functions, traces, services, APIs to easily identify performance bottlenecks, any edge case errors or issues.

Amazon Managed Service for Prometheus helps you manage monitoring and alerting for your containerized applications and infrastructure at scale. It is fully compatible with the popular upstream open-source Prometheus. You can collect and access performance and operational data from container workloads on AWS and on-premises. It simplifies the setup of Prometheus and automates the ongoing operations and maintenance.

Amazon Managed Grafana is a managed service for open-source Grafana. It simplifies interactive visualization and analysis for your data sources at scale. You can visualize, analyze, and alarm on your metrics, logs, and traces collected from multiple data sources in your observability system. This includes AWS services, third-party ISVs, and other resources in your environment.

AWS Distro for OpenTelemetry (ADOT) is a secure production-ready, AWS-supported distribution of the OpenTelemetry project, which provides a single set of open-source APIs, libraries, and agents to collect and correlate distributed traces and metrics. ADOT consists of SDKs, auto-instrumentation agents, collectors and exporters to send data to back-end services, including Amazon CloudWatch, X-Ray, and Amazon Managed Service for Prometheus.

Storage

By default, all files created inside a container write to a thin writable container layer. This data doesn't persist when the container no longer exists and it is tightly coupled to the host where a container is running. Amazon ECS supports the following data volume options for containers:

- **Bind mounts**: With a bind mount, a file or directory on a host is mounted onto one or more containers. For tasks hosted on Amazon EC2 instances, the data can be tied to the lifecycle of the host Amazon EC2 instance by specifying a host and optional `sourcePath` value in your task definition. Within the container, writes to the `containerPath` are persisted to the underlying volume defined in the `sourcePath` independently from the container's lifecycle. You can also

share data from a source container with other containers in the same task. Tasks hosted on AWS Fargate using platform version 1.4.0 or later receive by default a minimum of 20 GB of ephemeral storage for bind mounts, which can be increased to a maximum of 200 GB. Review the Amazon ECS developer guide for more use cases and considerations.

- **Docker Volumes**: With the support for Docker volumes, you can have the flexibility to configure the lifecycle of the Docker volume and specify whether it's a scratch space volume specific to a single instantiation of a task, or a persistent volume that persists beyond the lifecycle of a unique instantiation of the task. You can use volume drivers (also referred to as plugins) to integrate the volumes with external storage systems, such as Amazon EBS. To learn more, visit the Amazon ECS documentation.
- **Amazon EFS**: It provides simple, scalable, and persistent file storage for use with your Amazon ECS tasks. With Amazon EFS, storage capacity is elastic, growing and shrinking automatically as you add and remove files. Your applications can have the storage they need, when they need it. Amazon EFS volumes are supported for tasks hosted on Fargate or Amazon EC2 instances. For more information, see Amazon EFS volumes.

Amazon FSx for Windows File Server volumes is available for your Windows tasks on Amazon ECS with Amazon EC2 launch type. This provides fully managed Windows file servers backed by a Windows file system.

Kubernetes supports many types of volumes. Ephemeral volume types have a lifetime of a pod, but persistent volumes exist beyond the lifetime of a pod. When a pod ceases to exist, Kubernetes destroys ephemeral volumes; however, Kubernetes does not destroy persistent volumes. For any kind of volume in a given pod, data is preserved across container restarts. For Amazon EKS, the Container Storage Interface (CSI) allows exposing storage systems/backends to containerized workloads as persistent storage. CSI driver provides an interface to manage the lifecycle of Amazon EBS, Amazon EFS, and Amazon FSx for persistent volumes. For more information, see Kubernetes Volumes.

Networking

AWS container services take advantage of the native networking features of Amazon Virtual Private Cloud (Amazon VPC). This allows the hosts running your containers to be in different subnets, across Availability Zones, providing high availability.

Additionally, you can take advantage of VPC features like Network Access Control Lists (NACL) and Security Groups to ensure that only network traffic you want to allow to come in or leave your

container. For Amazon ECS, the main networking modes are ones that operate at a task level using the *awsvpc* network mode or the traditional bridge network mode which runs a built-in virtual network inside each Amazon EC2 instance, `awsvpc` is the only network available for AWS Fargate.

Amazon EKS uses Amazon VPC Container Network Interface (CNI) plugin for Kubernetes for the default native VPC networking to attach network interfaces to Amazon EC2 worker nodes. The VPC CNI plugin prioritizes pods in the VPC. The pods in an Amazon EKS cluster receive IP addresses from the private IP ranges of your VPC. When the number of pods running on the node exceeds the number of addresses that can be assigned to a single network interface, the VPC CNI plugin starts allocating a new network interface, if the maximum number of network interfaces for the instance aren't already attached. Using CNI custom networking, you can assign IP addresses from a different CIDR block than the subnet that the primary network interface is connected to. Amazon EKS also supports IPv6 networking. The VPC CNI plugin allows use of VPC flow logs for troubleshooting and compliance auditing and security groups for isolation and regulatory requirements.

VPC CNI also natively supports Kubernetes network policies, although you can also use third-party libraries such as Calico for this. This allows you to control network communication inside your Kubernetes cluster at a very granular level. In addition, VPC CNI allows pre-warming elastic network interfaces (ENIs) and increased IP addressing performance by modifying configurable parameters. For AWS Nitro based Amazon EC2 instances, it offers a prefix-delegation mode to increase pod density on a worker node. This is supported for both Linux and Windows nodes. Prefix delegation increases the number of IP addresses on the Elastic Network Interface (ENI) by assigning prefixes instead of individual secondary IP addresses.

Amazon EKS and Amazon ECS both support accessibility over private links. Using AWS PrivateLink ensures that calls to the Amazon ECS or Amazon EKS service stay within the Amazon network backbone and do not traverse the internet.

AWS container services like Amazon EKS and Amazon ECS now support IPv6 either through dual-stack or IPv6 only mode. Amazon EKS and AWS App Mesh support IPv6 in both dual-stack and IPv6-only mode where services like Amazon ECS and Fargate support IPv6 through dual-stack mode for now. IPv6 is growing adoption and customers using AWS container services can take advantage of this feature when running their workloads.

When you have your microservices deployed as containers, you want to establish service-to-service communication for exchanging information and implementing business flows. One way to achieve service-to-service communication is direct communication using service discovery. In this approach, you can use the AWS Cloud Map service discovery integration with Amazon ECS. Using service

discovery, Amazon ECS synchronizes the list of launched tasks to AWS Cloud Map, which maintains a DNS hostname that resolves to the internal IP addresses of one or more tasks from that service. In Amazon EKS, this is internally handled by Core-DNS component of the cluster. App Mesh is a service mesh that can help manage many services and have better control of how traffic gets routed among services. App Mesh functions as an intermediary between basic service discovery and load balancing. For serving ingress traffic into your Amazon EKS clusters, you can use AWS Load Balancer Controller to act as ingress controller and for setting up Application Load Balancers and Network Load Balancers.

Amazon ECS Service Connect is a feature in Amazon ECS that brings App Mesh-like capabilities in an integrated approach. Amazon ECS Service Connect provides service discovery, traffic resilience and observability out of the box for Amazon ECS-based containerized applications.

Amazon VPC Lattice is a recently launched application networking service and is ideal for connecting micro-services that are distributed across a mix of Amazon EKS and Kubernetes, native Amazon EC2/ASG, and serverless environments (Lambda and Fargate). VPC Lattice is best suited to customers who prefer the automation of service discovery, traffic-management, authentication, authorization, and observability across VPCs and accounts without having to deploy and operate sidecar-based service-meshes and prefer not requiring any prior VPC networking experience in deploying their modern application architectures.

Security

The shared responsibility of security applies to AWS container services as well. AWS manages the security of the infrastructure that runs your containers. However, controlling access for your users and your container applications is your responsibility as the customer. AWS Identity and Access Management (IAM) plays an important role in the security of AWS container services. The permissions provided by the IAM policies attached to the different principals in your AWS account, determines what capabilities they have. You should avoid using long-lived credentials such as access keys and secret access keys with your container applications. IAM roles provide you with temporary security credentials for your role session. You can use roles to delegate access to users, applications, or services that don't normally have access to your AWS resources. There are usually IAM roles at two different levels. The first determines what a user can do within AWS Container services, and the second is a role that determines which other AWS services your container applications running in your cluster can interact with. For Amazon EKS, the IAM roles works together with Kubernetes RBAC to control access at multiple levels. IAM roles for service accounts (IRSA) with Amazon EKS, enables you to associate an IAM role with a Kubernetes service

account. This service account can then provide AWS permissions to the containers in any pod that uses that service account.

With this feature, you no longer need to over-provision permissions to the IAM role associated with the Amazon EKS node, so that pods on that node can call AWS APIs.

Similarly, you can employ principles of least privilege by assigning granular IAM roles to Amazon ECS tasks. Assigning each Amazon ECS task a role aligns with the principle of least privileged access and allows for greater granular control actions and resources.

For your audit needs, you can use AWS CloudTrail, a service that provides a record of actions taken by a user, role, or another AWS service in AWS container services. Using the information collected by CloudTrail, you can determine the request made to Amazon ECS, the IP address from which the request was made, who made the request, when it was made, and additional details. For Amazon EKS, enabling control plane logging gives you the ability to trace calls to the Amazon EKS control plane. The audit logs are sent to Amazon CloudWatch Logs. You can configure the containers in your Amazon ECS tasks to send log information to CloudWatch Logs. For intelligent threat detection, Amazon GuardDuty EKS audit log monitoring and Amazon EKS runtime protection can be used to detect threat actors inside your Amazon EKS clusters.

AWS Secrets Manager and AWS Systems Manager Parameter Store are two services that can be used to secure sensitive data used within container applications. Systems Manager Parameter Store provides secure, hierarchical storage of data with no servers to manage. Secrets Manager provides additional capabilities that includes random password generation and automatic password rotation. Data stored within Systems Manager Parameter can be encrypted using AWS KMS and Secrets Manager uses it to encrypt the protected text of a secret as well. AWS container services can integrate with either Systems Manager Parameter Store or Secrets Manager to process sensitive data securely. Secrets Manager, however, also includes the ability to automatically rotate secrets, generate random secrets, and share secrets across AWS accounts. Kubernetes secrets enables you to store and manage sensitive information, such as passwords, docker registry credentials, and TLS keys using the Kubernetes API. Kubernetes secrets are, by default, stored as unencrypted base64- encoded strings. They can be retrieved in plain text by anyone with API access, or anyone with access to Kubernetes' underlying data store. You can apply native encryption-at-rest configuration provided by Kubernetes to encrypt the secrets at rest.

However, this involves storing the raw encryption key in the encryption configuration, which is not the most secure way of storing encryption keys. Kubernetes stores all secret object data within `etcd`, encrypted at the disk level using AWS-managed encryption keys. You can further encrypt

Kubernetes secrets using a unique data encryption key (DEK). You are responsible for applying necessary RBAC based controls to ensure that only the right roles in your Kubernetes cluster have access to the secrets and the IAM permissions for the AWS KMS key is restricted to authorized principals.

The container services take advantage of different constructs provided by Amazon VPC. By applying right controls for IP addresses and ports at different levels, you can ensure that only desired traffic enters and leaves your container applications. In Amazon EKS and Amazon EKS Anywhere, restricting network traffic between pods can be achieved by implementing network policies in your clusters. Network policies give you a mechanism to restrict network traffic between pods (often referred to as east/west traffic) and between pods and external services. You can now use Amazon VPC CNI to implement both pod networking and network policies to secure the traffic in their Kubernetes clusters. Alternately, you can also use security groups for pod feature in Amazon EKS and Fargate on Amazon EKS if you need to control communication between services that run within the cluster and service the run outside the cluster such as an Amazon RDS database. With security groups for pods, you can assign an existing security group to a collection of pods.

Encrypting network traffic prevents unauthorized users from intercepting and reading data when that data is transmitted across a network. For encrypting traffic in transit, you can use service mesh like App Mesh with Amazon ECS and Amazon EKS. With App Mesh, you can configure TLS connections between the Envoy proxies that are deployed with mesh endpoints. You can also use AWS PrivateLink to access your Amazon ECS and Amazon EKS clusters more securely without the need to go over the internet.

Container image is considered the first line of defense against an attack. An insecure, poorly constructed image can allow an attacker to escape the bounds of the container and gain access to the host. To prevent this from happening, you should create minimal container images or use distro-less images. Also, use multi-stage builds and linting tools in your Docker files. In addition, you can use Amazon ECR's basic and enhanced scanning features to continuously scan your container images for any security vulnerability. Also, use immutable tags with Amazon ECR. Immutable tags force you to update the image tag on each push to the image repository. This can prevent attackers from overwriting an image with a malicious version without changing the image's tags. Additionally, it gives you a way to easily and uniquely identify an image. Signing your container images that you store in Amazon ECR establishes authenticity and provenance, giving you the ability to determine if content comes from a particular party. You can use AWS Signer Container Image Signing capability for signing and verifying container images stored in Amazon ECR.

To implement stricter security controls across applications that run in your Amazon EKS cluster, use Policy-as-Code (PAC) solutions like Open Policy Agent (OPA) that provide guardrails to guide cluster users, and prevent unwanted behaviors, through prescribed and automated controls. Alternatively, you can also use Pod Security Standards (PSS) and Pod Security Admission (PSA), which come built into Kubernetes 1.25 onwards and are supported by Amazon EKS. While using Amazon EKS and Amazon ECS, restrict containers that can run with privileged access. Containers that run as privileged inherit all the Linux capabilities assigned to root on the host. Running a container as privileged isn't supported on Fargate.

For securing your infrastructure, consider using Amazon ECS optimized AMIs, Amazon EKS optimized AMIs, and Bottlerocket. They contain a minimal set of OS packages and binaries necessary to run your containerized workloads. They provide a smaller attack surface, which reduces the chances of your instances being compromised by questionable packages and modules. You can run Amazon Inspector to assess hosts for exposure, vulnerabilities, and deviations from best practices.

Build and deploy automation

Containers have become a feature component of continuous integration (CI) and continuous deployment (CD) workflows. Because containers can be built programmatically using Dockerfiles, they can be automatically rebuilt anytime a new code revision is committed. Containers promote the idea of immutable deployments. Each build creates a new set of container images. Each deployment is a new set of containers, and it’s easy to roll back by deploying containers that reference older images.

AWS Code Services in AWS Developer Tools provide convenient AWS native stack options to automate build and deployment for your applications. They have native integrations with the AWS containter services and provide tooling to pull the source code from the source code repository, build the container image, push the container image to the container registry, and deploy the image as a running container in one of the container services. AWS CodeBuild uses Docker images to provision the build environments, which makes it flexible to adapt to the needs of the application you are building. A build environment represents a combination of operating system, programming language runtime, and tools that CodeBuild uses to run a build. For code not containerized already, consider using AWS App2Container. App2Container is a command-line tool that can analyze and build an inventory of all .NET and Java applications running in virtual machines, on-premises or in the cloud. App2Container packages the application artifact and identified dependencies into container images, configures the network ports, and generates a

Dockerfile, Amazon ECS task definition, or Kubernetes deployment manifests by integrating with various AWS services.

Non-AWS tooling for CI/CD like GitHub, Jenkins, DockerHub and many others can also integrate with the AWS container services using the APIs and you can continue to use them. For Amazon EKS, GitOps provides a way to manage application and infrastructure deployment where the whole system is described declaratively in a Git repository. GitOps provides a set of best practices that unifies deployment, management, and monitoring for clusters and applications. Flux from Weaveworks and ArgoCD are two popular options for GitOps, although several other have started to emerge. All GitOps solutions integrate seamlessly with Amazon EKS and help you achieve CD through the declarative nature of Kubernetes manifests.

Infrastructure as code and platform automation

When you define your cloud resources as code, you spend less time creating and managing the infrastructure and achieve better operations through reusability and reduced changes of error. As with other AWS services, AWS CloudFormation provides you a way to model and set up your container resources. AWS CloudFormation templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. All AWS container services have AWS CloudFormation support, providing you with an option to script your container infrastructure on AWS. AWS CloudFormation is powerful in its capabilities, but if you're unfamiliar with JSON or YAML, AWS also provides other options to script your container environments.

AWS Copilot CLI is a tool for developers to build, release, and operate production-ready containerized applications on Amazon ECS and AWS Fargate. Copilot takes best practices from infrastructure to continuous delivery, and makes them available to customers from the comfort of their command line. You can also monitor the health of your service by viewing your service's status or logs, scale up or down production services, and spin up a new environment for automated testing. For Amazon EKS, *eksctl* is a simple CLI tool for creating and managing clusters on Amazon EKS. It uses AWS CloudFormation under the covers, but allows you to specify your cluster configuration information using a config file, with sensible defaults for configuration that is not specified. If you prefer to use a familiar programming language to define cloud resources, you can use AWS Cloud Development Kit (AWS CDK). CDK is a software development framework for defining cloud infrastructure in code and provisioning it through AWS CloudFormation. Today CDK supports TypeScript, JavaScript, Python, Java, C#/.Net, and (in developer preview) Go. Amazon EKS Blueprints helps you compose complete Amazon EKS clusters, fully bootstrapped

with the operational software that is needed to deploy and operate workloads. A blueprint defines configuration for the desired state of your Amazon EKS environment, such as the control plane, worker nodes, and Kubernetes add-ons. Blueprints can be implemented using Terraform or CDK.

Platform engineering is a discipline that promotes the idea of self-provisioning infrastructure on demand. This reduces friction within an organization and allows development teams to move in an agile manner. AWS Proton is a fully managed delivery service for container and serverless applications that enables platform engineering teams to connect and coordinate all the different tools needed for infrastructure provisioning, code deployments, monitoring, and updates.

Scaling

Amazon ECS is a fully managed container orchestration service with no control planes to manage scaling. Amazon ECS provides options to efficiently auto-scale Amazon EC2 cluster nodes and Amazon ECS services for your clusters. Amazon ECS will ensure the Amazon EC2 Auto Scaling groups scale in and out as needed with no further intervention required. Amazon ECS cluster auto-scaling relies on Amazon ECS capacity providers, which provide the link between your Amazon ECS cluster and the Auto Scaling groups you want to use. The core responsibility of Amazon ECS cluster auto scaling is to ensure that the right number of instances are running in an Auto Scaling group to meet the needs of the tasks assigned to that Group, including tasks already running as well as tasks the customer is trying to run that don't fit on the existing instances. For more details, read the Deep Dive on Amazon ECS Cluster Auto Scaling blog.

Amazon ECS Auto Scaling is the ability to automatically increase or decrease the desired count of tasks in your Amazon ECS service for both Amazon EC2 and Fargate based clusters. You can use the services' CPU and memory or other CloudWatch metrics. Amazon ECS Auto Scaling supports the following types of auto scaling:

- **Target Tracking Scaling Policies**: Increase or decrease the number of tasks that your service runs based on a target value for a specific metric.
- **Step Scaling Policies**: Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments that vary based on the size of the alarm breach.
- **Scheduled Scaling**: Increase or decrease the number of tasks that your service runs based on the date and time.

Amazon EKS automatically manages the availability and scalability of the Kubernetes control plane nodes, which consist of Kubernetes API server, kube-scheduler, kube-controller-manager, and etcd

nodes. Kubernetes provides following options, compatible with Amazon EKS, to scale worker nodes and pods:

- The Kubernetes **Cluster Autoscaler** automatically adjusts the number of worker nodes in your cluster so that all pods fail have a place to run and make sure there are no unnecessary worker nodes. Amazon EKS node groups are provisioned as part of an Amazon EC2 Auto Scaling group, which is compatible with the Cluster Autoscaler.
- The Kubernetes **Horizontal Pod Autoscaler** automatically scales the number of pods in a deployment, replication controller or stateful set based on CPU utilization or with custom metrics. This can help your applications scale out to meet increased demand or scale in when resources are not needed, thus freeing up your nodes for other applications, similar to Amazon ECS Auto Scaling. When you set a target CPU use percentage, the Horizontal Pod Autoscaler scales your application in or out to try to meet that target.
- KEDA is an open-source Kubernetes-based Event Driven Autoscaler that automatically scales the number of pods, and works alongside standard Kubernetes components like the Horizontal Pod Autoscaler. With KEDA, you can explicitly map the apps with various event sources that it supports to auto scale.
- The Kubernetes **Vertical Pod Autoscaler** frees the users from necessity of setting up-to-date resource limits and requests for the containers in their pods based on historical resource usage over time. By default, it only provides the calculated recommendation without automatically changing resource requirements of the pods, but when auto mode is configured, it will resize the pod requests automatically and restart existing pods onto nodes with the appropriate resource amounts. It will also maintain ratios between limits and requests that were specified in initial containers configuration.
- Karpenter is an open-source cluster autoscaler that automatically launches right-sized nodes in response to unschedulable pods without Amazon EC2 Auto Scaling groups. Karpenter evaluates the aggregate resource requirements of the pending pods and chooses the optimal instance type to run them. It works natively with Kubernetes scheduling constraints. It also supports a consolidation feature to help lower cluster compute costs by looking for opportunities to remove under-utilized nodes, replace expensive nodes with cheaper alternatives, and consolidate workloads onto more efficient compute resources.

Conclusion

Using containers on AWS can accelerate your software development by creating constructive collaboration between your development and operations teams. The efficient and rapid provisioning, the promise of build once, run anywhere, the separation of duties by a common standard, and the flexibility of portability that containers provide offer advantages to organizations of all sizes. AWS provides a range of purpose-driven services for containers and a range of complementary services, making it easy to get started with containers and run a variety of workloads using containers at scale.

Contributors

Contributors to this document include:

- Chance Lee, Solutions Architect, Amazon Web Services
- Sushanth Mangalore, Solutions Architect, Amazon Web Services
- Ratnopam Chakrabarti, Solutions Architect, Amazon Web Services

Further reading

For additional information, see:

- Container Migration Methodology
- Best Practices for writing Dockerfiles
- Introducing AWS App Runner
- IAM roles for Kubernetes service accounts
- Amazon EKS Networking
- Amazon ECS using AWS Copilot
- Amazon EKS Best Practices Guides
- Amazon ECS Best Practices Guide
- Amazon ECS Workshop
- Amazon EKS Workshop
- Data on EKS
- Amazon EKS Anywhere
- Amazon ECS Anywhere
- Karpenter documentation
- Choosing an AWS container service

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Whitepaper updated	Major revision to expand content and shift focus away from Docker	March 8, 2024
Whitepaper updated	Whitepaper updated for technical accuracy	July 26, 2021
Initial publication	Whitepaper first published.	April 1, 2015

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.