Docker on AWS

AWS Whitepaper

Docker on AWS: AWS Whitepaper

Table of Contents

Docker on AWS

Publication date: **July 26, 2021** (*Document history* (p. 17))

Running containers in the cloud

This whitepaper provides guidance and options for running Docker on AWS. Docker is an open platform for developing, shipping, and running applications in a loosely isolated environment called a container. Amazon Web Services (AWS) is a natural complement to containers and offers a wide range of scalable infrastructure services upon which containers can be deployed. You will find various options such as AWS Elastic Beanstalk, Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), AWS Fargate (Fargate), and AWS App Runner. This paper cover details of each option and key components of the container orchestration.

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

Prior to the introduction of containers, developers and administrators were often faced with the challenges of compatibility restrictions with applications workloads having to be built specifically for its pre-determined environment. If this workload needed to be migrated, for example from bare metal to a virtual machine (VM), or from a VM to the cloud, or between service providers, this typically meant rebuilding the application or the workload entirely to ensure compatibility with the new environment. Container was introduced to overcome these incompatibilities by providing a common interface. With the release of Docker, the interest in containers technology has rapidly increased.

Docker is an open-source project that uses several resource-isolation features of the Linux kernel to sandbox an application, its dependencies, configuration files, and interfaces inside of an atomic unit called a container. This allows a container to run on any host with the appropriate kernel components, while shielding the application from behavioral inconsistencies due to variances in software installed on the host. Containers use operating system level virtualization compared to VMs, which use hardware level virtualization using hypervisor, which 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 being isolated from neighboring containers. This layer of isolation allows consistency, flexibility, and portability that enable rapid software deployment and testing. There are many ways in which using containers on AWS can benefit your organization. Docker has been widely employed in use cases such as distributed applications, batch jobs, continuous deployment pipelines, and etc. The use cases for Docker continue to grow in areas like distributed data processing, machine learning, streaming media delivery and genomics.

The following examples show how AWS services can integrate with Docker:

- **Amazon SageMaker** provides pre-built Docker Images for Deep Learning through TensorFlow and PyTorch or lets you bring your custom pre-trained models through Docker images.

- **Amazon EMR on Amazon EKS** provides a deployment option to run open- source big data frameworks on Amazon EKS.
- **Bioinformatics applications for Genomics** within Docker containers on Amazon ECS provide a consistent, reproducible run-time environment.
- **For many SaaS providers**, the profile of Amazon EKS represents a good fit with their multi-tenant microservices development and architectural goals.

Container benefits

The rapid growth of Docker containers is being fueled by the many benefits that it provides. If you have applications that run on VMs or bare metal servers today, you should consider containerizing them to take advantage of the benefits that come from Docker containers. These benefits can be seen across your organization, from developers and operations, to Quality Assurance (QA). The primary benefits of Docker are speed, consistency, density, and portability.

Speed

Because of their lightweight and modular nature, containers can enable rapid iteration of your applications. Development speed is improved by the ability to deconstruct applications into smaller units. This reduces shared resources between application components, leading to fewer compatibility issues between required libraries or packages. Operational speed is improved, because code built in a container on a developer's local machine can be easily moved to a test server by simply moving the container. The container startup time primarily depends on the size of the container image, cache, and the time to pull the image and start the container on host. To improve the container startup time, you must keep the size of image as small as possible, using techniques like multi-stage builds and local cache when applicable. For more information, see Best practices for writing Dockerfiles.

Consistency

The consistency and fidelity of a modular development environment provide predictable results when moving code between development, test, and production systems. By verifying that the container encapsulates exact versions of necessary libraries and packages, it is possible to minimize the risk of bugs due to slightly different dependency revisions. 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 earlier. This practice helps avoid *drift* of packages across a fleet of containers, versions of your application, or development, test, and production 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 system. Resource efficiency is a natural result of the isolation and allocation techniques that containers use. Containers can easily be restricted to a certain number of CPUs and allocated specific amounts of memory. By understanding what resource a container needs and what resource is available to your VM or underlying host server, it's possible to maximize the containers running on a single host, resulting in higher density, increased efficiency of compute resources, and less wastage on excess capacity. Amazon ECS achieves this through placement strategies. The *binpack* placement strategy tries to optimize placement of containers to be as 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. This further enables you to efficiently utilize resources by verifying that your tasks are running on instance types suitable for your workload, by logically separating your tasks using task groups.

Amazon EKS uses the native Kubernetes scheduling and placement strategy, which tries to place pods on nodes to match the requirements of your workloads across nodes and not to place pods on nodes where there aren't sufficient resources.

Kubernetes allows you to limit the resources like CPU and memory to Kubernetes namespaces, pods, or containers. For more information, see Scheduling (p. 8).

Portability

The flexibility of Docker containers is based on their portability, ease of deployment, and smaller size compared to virtual machines. Like Git, Docker provides a simple mechanism for developers to download and install Docker containers and their subsequent applications using the command `docker pull`. Because Docker provides a standard interface, it makes containers easy to deploy wherever you like, providing portability among different versions of Linux, your laptop, or the cloud. The images Docker builds are compliant with OCI (Open Container Initiative), which was created to support fully interoperable container standards. Docker can build images by reading the instructions from a *Dockerfile*, which is a text-based manifest. You can run the same Docker container on any supported version of Linux if you have the Docker stack installed on the host. Additionally, Docker supports Windows containers which can run on supported Windows versions. Containers also provide flexibility by making a microservices architecture possible. In contrast to common infrastructure models in which a virtual machine runs multiple services, packaging services inside their own container on top of a host OS allows a service to be moved between hosts, isolated from failure of other adjacent services, and protected from errant patches or software upgrades on the host system. Because Docker provides clean, reproducible, and modular environments, it streamlines both code deployment and infrastructure management. Docker offers numerous benefits for a variety of use cases, whether in development, testing, deployment, or production.

Containers orchestrations on AWS

Amazon Web Services (AWS) is an elastic, secure, flexible, and developer-centric ecosystem that serves as an ideal platform for Docker deployments. AWS offers the scalable infrastructure, APIs, and SDKs that integrate tightly into a development lifecycle and accentuate the benefits of the lightweight and portable containers that Docker offers to its users. In this section, we will discuss the different possibilities for container deployments using AWS services such as, AWS Elastic Beanstalk, Amazon Elastic Container Service, Amazon Elastic Kubernetes Service, AWS Fargate, and other additional services.

- **AWS Elastic Beanstalk** supports the deployment of web applications from Docker containers. With Docker 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. By using Docker with Elastic Beanstalk, you have an infrastructure that handles all the details of capacity provisioning, load balancing, scaling, and application health monitoring. Elastic Beanstalk can deploy a Docker image and source code to EC2 instances running the Elastic Beanstalk Docker platform. The platform offers multi-container (and single-container) support. You can also leverage the Docker Compose tool on the Docker platform to simplify your application configuration, testing, and deployment. In situations where you want to use the benefits of containers and want the simplicity of deploying applications to AWS by uploading a container image, AWS Elastic Beanstalk may be the right choice. While it is useful for deploying a limited number of containers, the way to run and operate containerized applications with more flexibility at scale is by using Amazon ECS.
- **Amazon ECS** is a fully managed container orchestration service and the easiest way to rapidly launch thousands of containers across a broad range of AWS compute options, using your preferred CI/CD and automation tools. Amazon ECS with EC2 launch mode provides an easy lift for your applications that run on VMs. The powerful simplicity of Amazon ECS enables you to grow from a single Docker container to managing your entire enterprise application portfolio across availability zones in the cloud and on-premises using Amazon ECS Anywhere, without the complexity of managing a control plane, add-ons, and nodes. Amazon ECS Clusters are made up of Container Instances, which are Amazon EC2 instances running the Amazon ECS container agent, which communicates instance and container state information to the cluster manager, and pre-configured *dockerd*, the Docker daemon. The Amazon ECS container agent is included in the Amazon ECS-optimized AMI, but you can also install it on any EC2 instance that supports the Amazon ECS specification. Your containers are defined in a task definition that you use to run individual tasks or tasks within an Amazon ECS service, that 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 various parameters such as the Docker image to use, which ports should be open, amount of CPU and memory to use with each task or container within a task and the IAM role the task should use. For more in-depth information about Amazon ECS Task and Service use-cases see the the Scheduling (p. 8) section under Key components (p. 8).
- **Amazon EKS** provides a natural migration path if you are using Kubernetes already and want to continue to make use of those skills on AWS for your container applications. 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 highly-available and secure clusters and automates key tasks such as 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 that are responsible for scheduling containers, managing the availability of applications, storing cluster data and other key tasks. Amazon EKS runs upstream Kubernetes, certified conformant for a predictable experience. You can easily migrate any standard Kubernetes application to Amazon EKS without needing to refactor 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. Amazon EKS Anywhere is a deployment option that enables you to easily create and operate Kubernetes clusters on-premises, including on your own virtual machines

and bare metal servers. Amazon EKS Anywhere provides an installable software package for creating and operating Kubernetes clusters on-premises and automation tooling for cluster lifecycle support. As new Kubernetes versions are released and validated for use with Amazon EKS, we will support three stable Kubernetes versions as part of the update process at any given time. The container runtime used in Amazon EKS clusters may change in the future but your Docker containers will still work and you shouldn't notice it. Amazon EKS will eventually move to containers as the runtime for the Amazon EKS optimized Amazon Linux 2 AMI. You can follow the containers roadmap issue for more details.

- **AWS Fargate** provides a way to run containers in a serverless manner with both Amazon ECS and Amazon EKS. AWS Fargate allows you to deliver autonomous container operations, which reduces the time spent on configuration, patching, and security. AWS 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 AWS 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. When you run your Amazon ECS tasks and services 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, AWS Fargate integrates with Kubernetes using controllers that are built by AWS using the extensible 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.
- **AWS App Runner** is a fully managed service that makes it easy to quickly deploy containerized web applications and APIs, at scale without any prior experience of running infrastructure on AWS. You can go from an existing container image, container registry, source code repository, or existing CI/CD workflow to a fully running containerized web application on AWS in minutes. AWS App Runner supports full stack development, including 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. AWS App Runner is ideal for you if you want to run and scale your application on AWS, without configuring or managing infrastructure services. This means, you will not have any orchestrators to configure, build pipelines to set up, load balancers to optimize, or TLS certificates to rotate. This really makes it the simplest way to build and run your containerized web application in AWS.
- **Other options:** There are additional Docker-specific offerings available on AWS, which can be useful based on the nature of your workloads. It is beyond the scope of this whitepaper to look at these offerings in detail, but we have extensive official AWS documentation and blog posts available for each of these offerings.
  - **AWS App2Container (A2C)** is a command-line tool which can analyze and build an inventory of all .NET and Java applications running in virtual machines, on-premises or in the cloud. A2C 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 YAML by integrating with various AWS services.
  - **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.
  - **AWS Batch** helps you to run batch computing workloads on the AWS Cloud. You can define job definitions that specify which Docker container images to run your jobs, which run as containerized applications on AWS Fargate or Amazon EC2 resources in your compute environment.
  - **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.
  - **Red Hat OpenShift Service on AWS (ROSA)** can accelerate your application development process if you are presently running Docker 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.

- **AWS Proton** is a fully managed delivery service for container and serverless applications for Platform engineering teams to connect and coordinate all the different tools needed for infrastructure provisioning, code deployments, monitoring, and updates.

Your choice is usually driven by how much control you want to retain, at the expense of additional management effort, versus how much AWS can manage for you in the environment the containers run in. For most use cases, you may want to consider starting on the fully-managed end of the spectrum (App Runner or Fargate) and work backwards towards more of a self-managed experience based on the demands of your workload. The self-managed experience can go to the extent of managing Docker containers on Amazon EC2 VMs without the use of any AWS managed services, so you have the flexibility to pick the orchestration solution that works best for your needs.

Key components

Container-enabled AMIs

AWS has developed a streamlined, purpose-built operating system for use with Amazon Elastic Container Service. The Amazon ECS-optimized AMI, built on top of Amazon Linux 2, is pre-configured with the Amazon ECS container agent, a Docker daemon with Docker runtime dependencies, which is the simplest way for you to get started and to get your containers running on AWS quickly. The Amazon EKS-optimized Amazon Linux AMI is also built on top of Amazon Linux 2, configured to work with Amazon EKS and it includes Docker, kubelet, and the AWS IAM Authenticator. Although you can create your own container instance AMI that meets 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 the Bottlerocket, a Linux-based open-source operating system purpose-built by AWS for running containers. It includes only the essential software required to run containers, and focuses on security and maintainability, providing a reliable, consistent, and safe platform for container-based workloads.

Scheduling

When applications are scaled out across multiple hosts, it becomes important to have the ability to manage each host node, Docker containers, and abstract away the complexity of the underlying platform. 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 orchestration 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. The service scheduler and the Run Task.
  - The service scheduler is suited for long running stateless applications that ensures an appropriate number of tasks are constantly 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 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. Alternatively, you can customize how the scheduler places tasks using task placement strategies and constraints.
- **Amazon EKS**: Kubernetes scheduler (*kube-scheduler*) becomes 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 do a reasonable placement but there are some circumstances where you may 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.
  - **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 cannot be scheduled, the scheduler tries to preempt (evict) lower priority pods make scheduling of the pending pod possible.
- **Lambda** is serverless so you don't need to manage where or how to scheduler your containers. After you create a container image in the Amazon ECR, you can select create and run the Lambda function.
- **Elastic Beanstalk** can deploy a Docker image and source code to Amazon EC2 instances running the Elastic Beanstalk Docker platform. Compared to Amazon EKS or Amazon ECS, Elastic Beanstalk's container scheduling features are less for the sake of the managed infrastructure provisioning. For more information on samples and help getting started with a Docker environment, see Using the Docker platform.

Container repositories

Docker containers are distributed in the form of Docker images. Docker images are a compile time construct, defined by the Dockerfile manifest, with a set of instructions to create the containers. Docker images are stored in container registries for delivery to applications that need them. Within a registry, a collection of related images is grouped together as repositories. Amazon Elastic Container Registry (Amazon ECR) is the AWS native managed container registry for Open Container Initiative (OCI) images, which provides a convenient option with native integration to the AWS ecosystem. With ECR, you can share container images privately within your organization using a private repository, by default only accessible within your AWS account by users with the necessary permissions. 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. 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. For developers starting out with containers, DockerHub API limits 100 image requests every six hours for anonymous usage, but with Amazon ECR public you get one unauthenticated pull every second, providing a less restrictive option to get started. Your limits increase significantly when you authenticate to Amazon ECR and this is the recommended way to work with container registries as your adoption increases.

Logging and monitoring

Treating logs as a continuous stream of events instead of 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 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 Docker and 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. During AWS re:Invent 2020, AWS launched Amazon Managed Service for Prometheus (AMP) and Amazon Managed Service for Grafana (AMG), two new open source-based managed services providing additional options to choose from. AWS also provides the option to discover and ingest Prometheus custom metrics to CloudWatch Container Insights to reduce the number of monitoring tools.

Given the pace at which new services and features are being launched in this space, AWS launched the One Observability Demo Workshop to help customers to get hands- on experience with AWS instrumentation options and the latest capabilities of AWS observability services in a self-paced, guided, sandbox environment.

Storage

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

- **BindMounts**: A file or directory on a host can be mounted into one or more containers. For tasks hosted on Amazon EC2, the data can be tied to the lifecycle of the host, by specifying a host and optional `sourcePath` value in your task definition. Within the container, writes toward 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. For tasks hosted on AWS Fargate using platform version 1.4.0 or later, they receive a minimum of 20 GB of ephemeral storage for bind mounts which can be increased to a maximum of 200 GB.
- **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.
- **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.

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, Container Storage Interface (CSI) driver provides a CSI interface to manage the lifecycle of Amazon EBS, Amazon EFS, Amazon FSx for Lustre for Persistent Volume. 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. Amazon VPC network policies restrict traffic between control plane components to within a single cluster. Control plane components for a cluster can't view or receive communication from other clusters or other AWS accounts, except as authorized with Kubernetes RBAC policies. The pods 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 plugin starts allocating a new network interface, if the maximum number of network interfaces for the instance aren't already attached. Using CNI customer networking, you can assign IP addresses from a different CIDR block than the subnet that the primary network interface is connected to. You also have the option to set network policies through third-party libraries for Calico, so you have the options to control network communication inside your Kubernetes cluster at a very granular level. More details on Amazon EKS networking are available in the AWS documentation.

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.

Other aspects of security are network security, audit capability, and secrets management. The container services take advantage of different constructs provided by Amazon VPC. By applying the right controls for IP addresses and ports at different levels, you can ensure that only desired traffic enters and leaves your container applications. 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.

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 use process sensitive data securely. 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.

CI/CD

Containers have become a feature component of continuous integration (CI) and continuous deployment (CD) workflows. Because containers can be built programmatically using Dockerfiles, containers can be automatically rebuilt anytime a new code revision is committed. Immutable deployments are natural with Docker. Each deployment is a new set of containers, and it's easy to rollback by deploying containers that reference previous images. AWS container services provide APIs that make deployments easy by providing the complete state of the cluster and the ability to deploy containers using one of the built-in schedulers or a custom scheduler.

AWS Code Services in AWS Developer Tools provide a convenient AWS native stack to perform CI/CD for your container applications. It provides 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. Non-AWS tooling for CI/CD like GitHub, Jenkins, DockerHub and many others can also integrate with the AWS container services using the APIs.

Infrastructure as code

You should define your cloud resources as code, so that you can spend less time creating and managing the infrastructure. As with other AWS services, AWS CloudFormation provides you a way to model and set up your container resources formatted text files in JSON or YAML describing the resources that you want to provision. 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. Alternately, if your organization already uses Terraform or similar tools that have modules for AWS container services, you can use them to define your infrastructure as code too.

Scaling

Amazon ECS is a fully managed container orchestration service with no control planes to manage scaling at all. Amazon ECS provides options to auto scale container instances and Amazon ECS services. Amazon ECS cluster auto scaling (CAS) enables you to have more control over how you scale the Amazon EC2 instances within a cluster. The core responsibility of CAS to ensure that the right number of instances are running in an Auto Scaling Group to meet the needs of the tasks including tasks already running as well as tasks the customer is trying to run that don't fit on the existing instances.

Amazon ECS Service 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 service CPU and memory utilization or other CloudWatch metrics. Amazon ECS Service Auto Scaling supports target tracking, step scaling and scheduled scaling policies. For more information, see Service auto scaling.

Amazon EKS automatically manages the availability and scalability of the Kubernetes control plane nodes. Amazon EKS supports following Kubernetes auto scaling options.

- **Cluster Autoscaler** automatically adjusts the number of worker nodes in your cluster when pods fail or are rescheduled onto other nodes. Amazon EKS node groups are provisioned as part of an Amazon EC2 Auto Scaling group, which are compatible with the Cluster Autoscaler.
- **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 Service Autoscaling.
- **Vertical Pod Autoscaler** frees the users from necessity of setting up-to-date resource limits and requests for the containers in their pods. By default, it provides the calculated recommendation without automatically changing resource requirements of the pods but when auto mode is configured, it will set the requests automatically based on usage and thus allow proper scheduling onto nodes so that appropriate resource amount is available for each pod. It will also maintain ratios between limits and requests that were specified in initial containers configuration.

For more information on large clusters, see considerations for large clusters.

Conclusion

Using Docker containers in conjunction with AWS can accelerate your software development by creating synergy 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. By providing a range of services that support containers along with an ecosystem of complimentary services, AWS makes it easy to get started with containers while providing the necessary tools to run containers at scale.

Contributors

Contributors to this document include:

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

Further reading

For additional information, see:

- Container Migration Methodology
- Best Practices for writing Dockerfiles
- Deploying AWS Elastic Beanstalk Applications from Docker Containers
- Introducing AWS App Runner
- Twelve-Factor Apps using Amazon ECS and AWS Fargate
- Blue/Green deployment with CodeDeploy
- IAM roles for Kubernetes service accounts
- Amazon EKS Networking
- Amazon ECS using AWS Copilot
- Amazon EKS Best Practices Guides
- Amazon ECS Workshop
- Amazon EKS Workshop

Document history

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

Change	Description	Date
Whitepaper updated (p. 17)	Whitepaper updated for technical accuracy	July 26, 2021
Initial publication (p. 17)	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 General Reference*.