Security Overview of AWS Fargate

First published April, 2022

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.

Contents

Abstract 1

Terminology 1

Introduction 1

Benefits of Fargate 2

Container orchestration and Fargate 3

Fargate Platform Versions 3

Security considerations with Fargate 4

Shared Security Responsibility Model 6

Fargate Control Plane and Data Plane 8

Fargate on EC2 and Firecracker 8

Fargate Control Plane 8

Fargate Data Plane 9

Security in Fargate 12

Isolation by Design 12

Network Security 13

Patching and security updates 14

Storage Security 15

Fargate Compliance 15

Fargate Security Partners 17

Conclusion 17

Contributors 18

Further Reading 18

Document revisions 18

Abstract

This paper is intended for existing and potential Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS) customers that choose to run their containerized workloads on AWS Fargate. It provides a security overview of Fargate, which is helpful for new adopters and deepens understanding of Fargate for current customers.

This white paper aims to inform customers running Linux workloads on Fargate using ECS or EKS. It's intended audience is anyone interested in understanding the foundations of Fargate.

Terminology

This document applies to workloads managed by Amazon ECS and Amazon EKS, therefore, we have preferred using generic terms to avoid overusing container orchestrator-specific parlance. For example, the concept of a task in Amazon ECS is very similar to that of a pod in Amazon EKS. At a high level, they both refer to a container or a group of container replicas. We have used both the terms task and pod as much as the flow of the text permitted, but unless specifically called out, the terms task and pods are interchangeable in this document.

We have used the term *host* to refer to the virtual machine that runs containers. Unless specifically called out, a *cluster* can mean either an Amazon ECS or an Amazon EKS cluster.

Introduction

Fargate is a serverless compute engine for Amazon ECS and Amazon EKS. Fargate allows developers and engineers to focus on their business capabilities while managing the underlying compute infrastructure on their behalf. It handles operational tasks such as provisioning, configuration, and scaling of compute infrastructure required to run containerized workloads. Fargate runs your containers on a highly available compute infrastructure and performs the administration of compute resources, including server and operating system maintenance, code and security updates, and automatic data plane scaling.

Fargate helps you improve security by isolating workloads by design. It uses hardware virtualization to run each task or pod in its own kernel, providing each task and pod an isolated compute environment. These are the reasons why AWS customers like Vanguard, Accenture, Square, Ancestry, and Vodafone have chosen to run their mission-critical applications on Fargate.

Benefits of Fargate

AWS Fargate is a serverless, pay-as-you-go compute engine that lets you focus on building applications without managing servers.

Fargate offers the following key benefits:

- **Isolation by design:** Fargate runs every task and pod in a dedicated virtual machine environment. Workloads running on Fargate have their own isolation boundary and do not share the underlying kernel, CPU and memory resources, or local storage with another task or pod. Fargate is architected with a secure approach to running containerized workloads that reduces the blast radius of attacks by isolating tasks and pods. If a vulnerable pod or task gets compromised, the tightened security boundary of Fargate is designed to prevent attackers from controlling other pods or tasks in the cluster.
- **Simplified data plane scaling:** Fargate helps you automatically scale the data plane as ECS and EKS scale your applications. Kubernetes add-ons such as Cluster Autoscaler that automate data plane scaling are not required with Fargate.
- **Up-to-date:** Fargate customers don't have to install and maintain components such as container runtime, kubelet, and kube-proxy. Fargate workloads run on newly provisioned hosts that use up-to-date compatible runtime components. Fargate workloads run on a newly created virtual machine, one that has not run any other task or pod. Fargate customers are not responsible for creating and updating the underlying host. Once a task or pod stops, Fargate terminates the virtual machine.
- **Resource-based pricing**: With Fargate, you pay only for the compute, storage, and network resources reserved by your containerized applications. It offers granular, per-second billing with flexible configuration and pricing options. Customers can further reduce their spend on Fargate by utilizing Fargate Spot and Compute Savings Plans.

Container orchestration and Fargate

Container orchestrators manage the operation of containerized workloads on the user's behalf. AWS offers customers a choice of two managed container orchestrators: Amazon ECS and Amazon EKS. Container orchestrators create, terminate, scale containers, and provide networking support for services to interconnect. The implementation of these operations depends on the orchestrator — for example, the procedure to scale workloads may differ in ECS and EKS — but the eventual result is conceptually identical. Both ECS and EKS help you autoscale workloads by adding or removing containers to maintain an application's performance as its workload fluctuates.

Fargate is the underlying serverless compute engine for ECS and EKS and AWS Batch; it provides Batch, ECS, and EKS compute infrastructure to run containerized workloads. In this context, Fargate is similar to Amazon EC2. Customers orchestrate containers using Batch, ECS, or EKS and use Fargate or EC2 to run containers. Customers never interact directly with Fargate but through ECS, EKS or Batch. Fargate assumes the responsibility of maintaining the underlying infrastructure that runs your containers; the container orchestrator manages container creation, execution, and termination.

Fargate Platform Versions

For ECS, Fargate Platform Versions (PV) refer to a specific runtime environment for the Fargate task infrastructure, comprised of a combination of the kernel and container runtime versions. New platform versions and platform version revisions are released as the runtime environment evolves, such as kernel or operating system updates, new features, bug fixes, or security updates.

Customers receive a task retirement notice when AWS determines that a security or infrastructure update is needed for a Fargate task. For more information, please see AWS Fargate task maintenance.

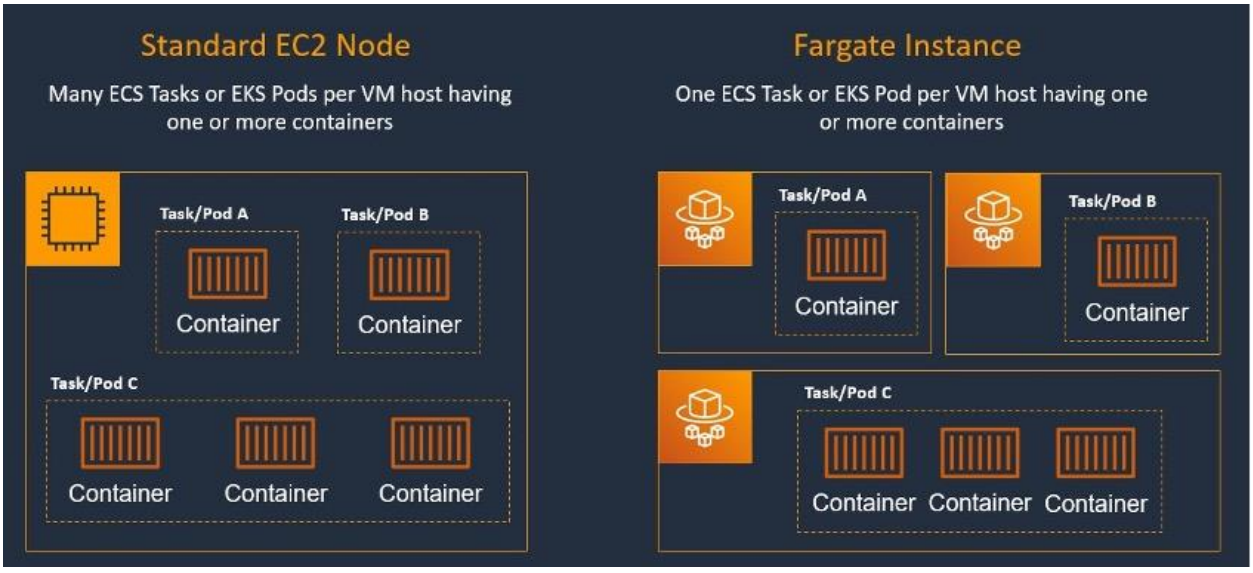
For EKS, customers use the latest Fargate Platform Version and revision for new deployments. Existing workloads move to the latest PV when redeploying the pods. Customers with long-running pods will want stop their pods periodically to ensure their pods use the latest Fargate PV revision.

Security considerations with Fargate

Fargate runs each workload in an isolated hardware virtualized environment. As a result, each pod or task gets dedicated infrastructure capacity. Containerized workloads running on Fargate do not share network interfaces, Fargate ephemeral storage, CPU, or memory with other tasks or pods. Customers can run multiple containers within a task or pod, including application containers and sidecar containers, or simply *sidecars*.

A sidecar is a container that runs alongside an application container in a Kubernetes pod or an ECS task. While the application container runs core application code, processes running in sidecars can augment the application. Sidecars help you segregate applications functions into dedicated containers, making it easier for you to update parts of your application.

Containers that are part of the same pod or task share resources when running on Fargate, as these containers will always run on the same host and share compute resources. Such containers can also share the ephemeral storage that Fargate provides. Linux containers in a task or pod share network namespaces, including the IP address and network ports. Inside a pod or a task, containers that belong to the pod or task can inter-communicate over the local loopback interface.



Fargate runs one task or pod per VM

The security hardened runtime environment in Fargate disallows using certain orchestrator features that are supported on EC2 instances. Following are the security

considerations customers should be aware of when architecting workloads that run on Fargate:

No privileged containers or access. Use cases such as running Docker in Docker require containers to be run in privileged mode. Features such as privileged containers or access are currently unavailable on Fargate.

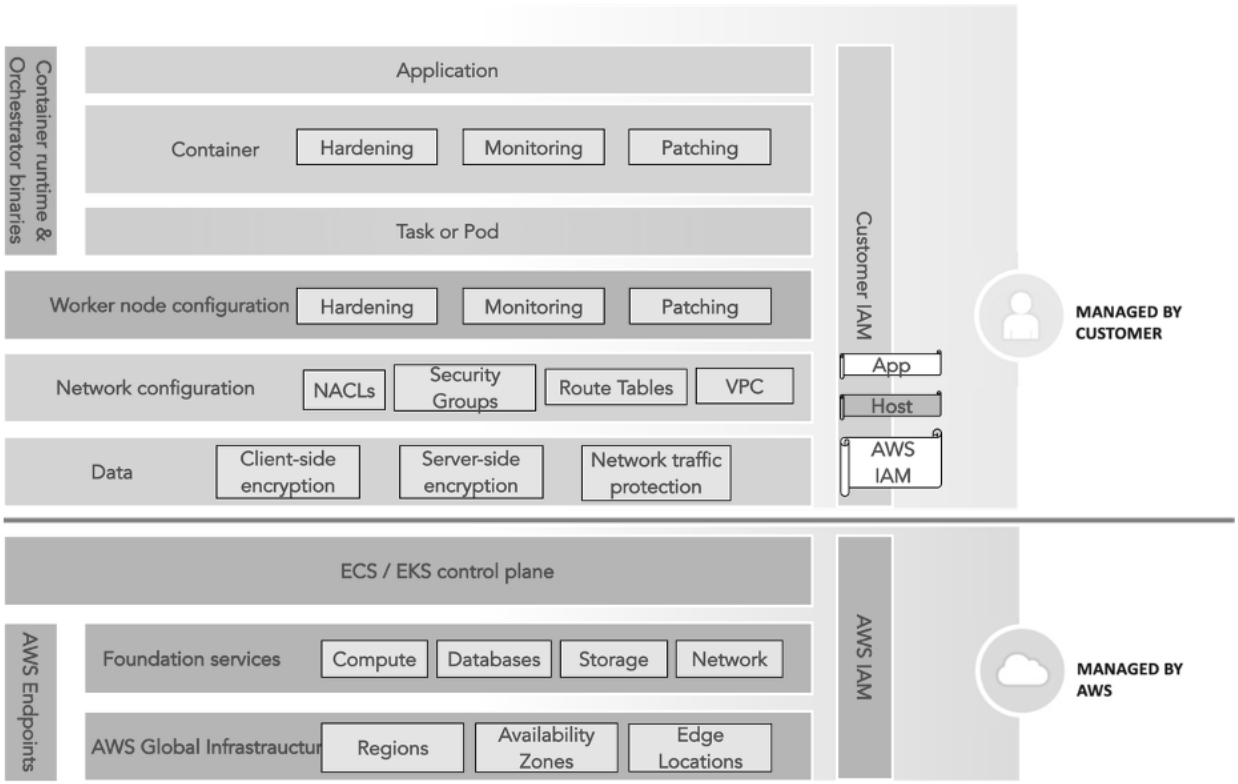
Limited access to Linux capabilities. The execution environment in which containers run in Fargate is restrictive to prevent container breakouts. ECS on Fargate supports adding `CAP_SYS_PTRACE` Linux capability, allowing observability tools like Sysdig Falco for workloads running on Fargate. Workloads including third party solutions such as Istio that require additional Linux capabilities are currently unsupported on Fargate. The list of capabilities Fargate workloads run with is available here.

No access to the underlying host. Neither customers nor AWS operators can connect to a host running customer workloads. Customers can use `kubectl exec` for EKS or ECS Exec for ECS to run commands in or get a shell to a container running on Fargate. This feature is helpful with collecting diagnostic information and debugging. Additionally, Fargate prevents containers from accessing underlying host's resources, such as the filesystem, devices, networking, container runtime, etc.

Networking. Fargate customers can use security groups and network ACLs to control inbound and outbound traffic. Fargate pods or tasks receive an IP address from the configured subnet in your VPC.

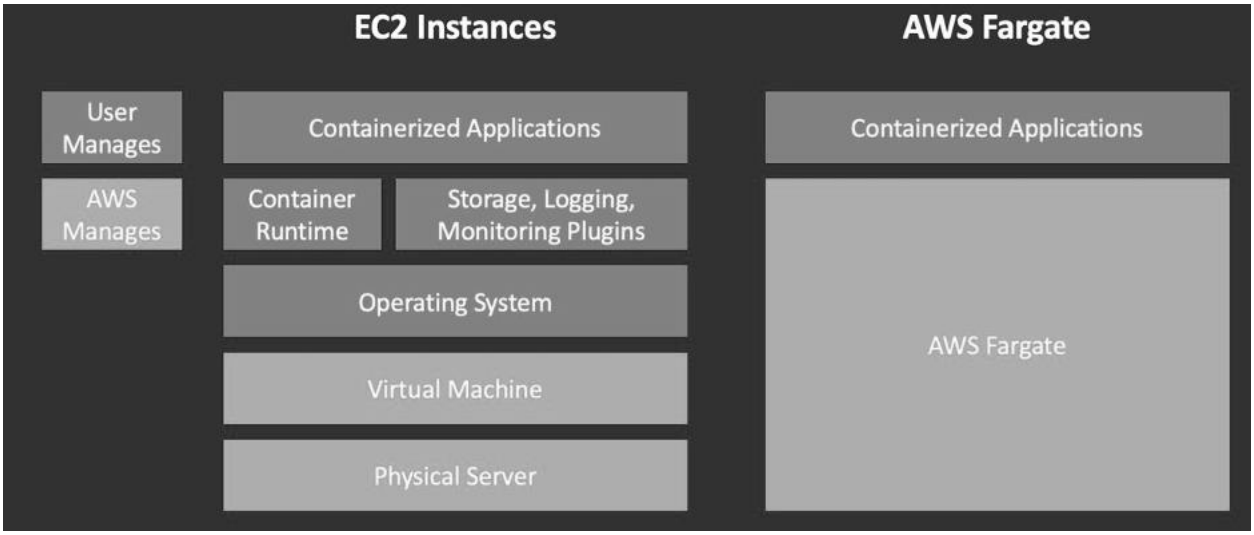
Shared Security Responsibility Model

Security and Compliance is a shared responsibility between AWS and the customer. This model reduces the customer's operational burden as AWS operates, manages, and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the service operates.



Shared Responsibility Model for Fargate

Fargate reduces your team's operational burden as AWS assumes the responsibility of managing the underlying Fargate infrastructure, operating system, and execution environment.



Shared Responsibility Model differences between EC2 and Fargate. Customers are responsible for securing their workloads when using Fargate.

When using Fargate, customers are not responsible for securing the compute infrastructure that runs their containers. Fargate will provision and patch the infrastructure upon which customer workloads run. Please see the Patching and security updates section for more details.

Fargate Control Plane and Data Plane

Fargate on EC2 and Firecracker

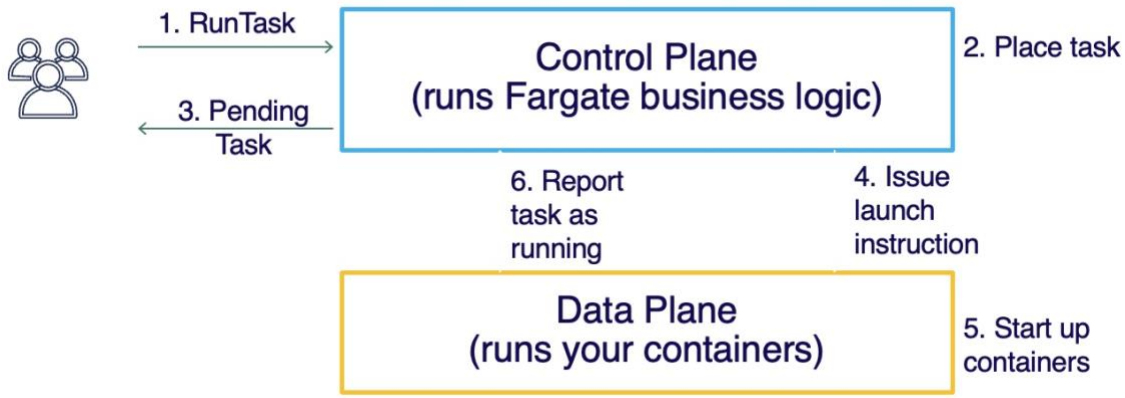
Fargate provides a managed data plane for customers to run ECS and EKS workloads. Fargate uses both EC2 instances and Firecracker micro-VMs running on EC2 bare-metal instances to run tasks and pods. The next section explains EC2 and Firecracker Control Plane and Data Plane.

Firecracker

Firecracker runs workloads in lightweight virtual machines, called microVMs, which combine the security and isolation properties provided by hardware virtualization technology with the speed and flexibility of containers. Firecracker is built on KVM, the same hypervisor on which EC2 Nitro instances are based. Hardware virtualization is designed to securely run tasks from different customers on the same physical server. Firecracker is a Virtual Machine Monitor (VMM) and not a full hypervisor.

For more information on Firecracker refer to the launch blog and https://firecracker-microvm.github.io/.

Fargate Control Plane



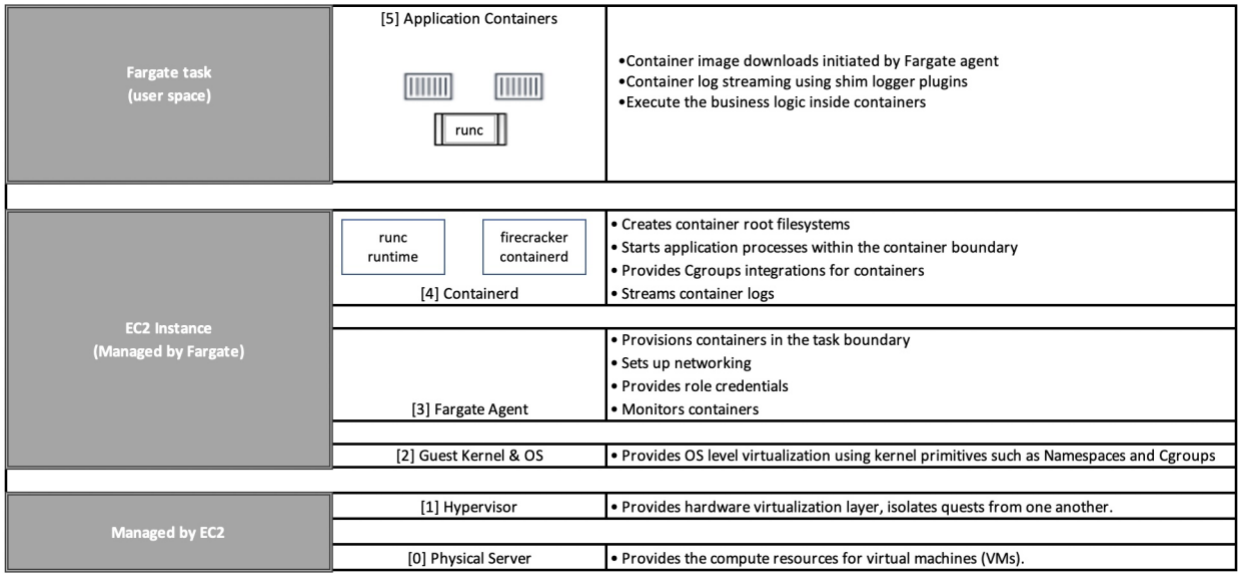
Fargate Control Plane

The preceding diagram gives a high-level overview of the interaction between the Control Plane and the Data Plane where a RunTask API call is made to start a new task using a specified task definition. The Control Plane consists of the services that are

running the Fargate business logic to orchestrate the task launches. The Data Plane consists of the fleet of services and underlying infrastructure that runs the container workloads using EC2 instances and Firecracker. These two data planes are described in further detail in the next section. The choice of which of the two data planes a task or pod is placed on is managed by Fargate, and customers do not need to configure this.

As shown in the preceding figure, when the ECS RunTask API call is made, ECS calls the Fargate control plane to place the task on appropriate compute based on task/pod parameters such as the requested number of vCPUs and amount of memory. Asynchronously, the Control Plane issues the instruction to launch the container in your task definition within the reserved server capacity. The status of this container in the Data Plane is then reported to the Control Plane.

Fargate Data Plane



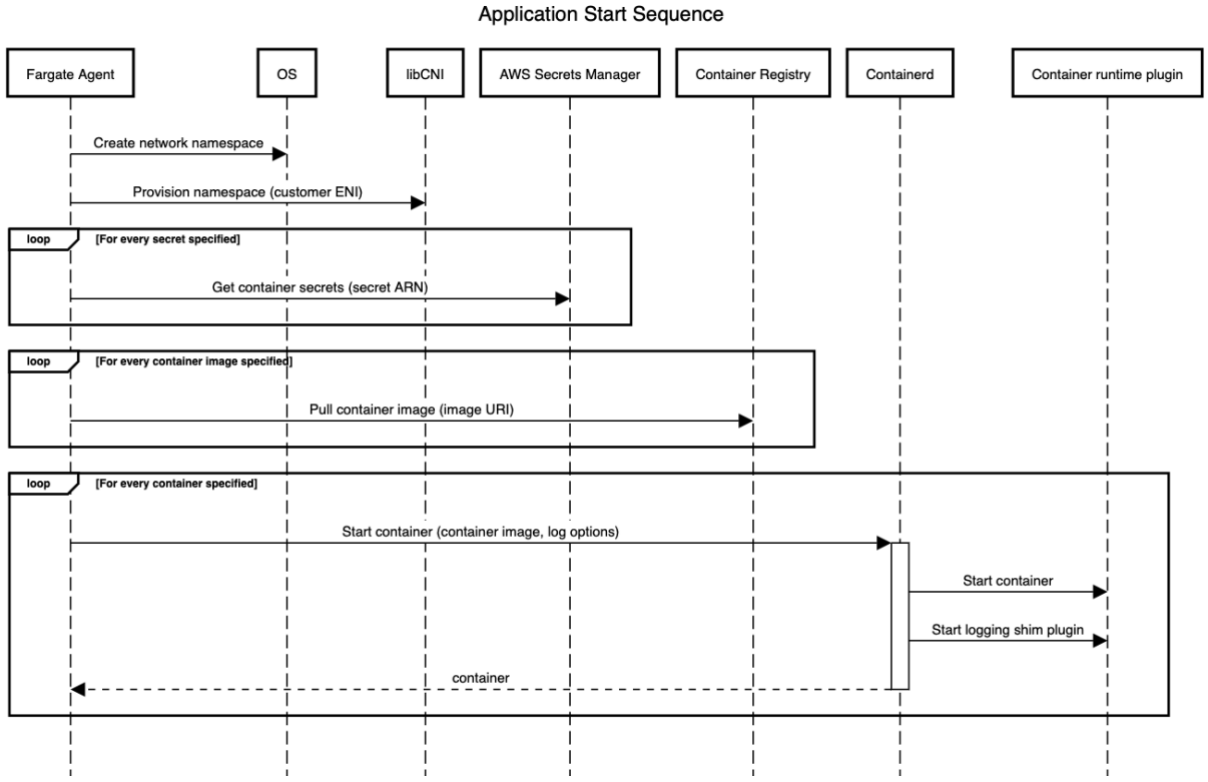
Fargate Data Plane

The EC2 instances used by Fargate are fully managed by AWS and run in AWS accounts and VPCs that Fargate owns and are invisible to customers. When a customer task is placed on an instance, an ENI is provisioned that connects the instance to the VPC designated by the customer. The instance runs a slimmed-down Amazon Linux 2 as the guest OS, with a Fargate Agent installed to communicate with the control plane. The container runtime is used to launch the container workloads. The primary network interface is in the Fargate VPC, where the instance is launched. When the instance gets

selected to host a task, an ENI is created within customer VPC and attached to the instance as a secondary network interface.

Additionally for Firecracker, the firecracker-containerd plugin is responsible for running containers within a Firecracker virtual machine (VM). Firecracker VMs do not execute as root on the host. Seccomp filters limit the host system calls Firecracker can use. The default filters only allow the bare minimum set of system calls and parameters that Firecracker needs to function correctly. Another jailer process can start the Firecracker process. You can find additional information in the Firecracker open-source design documentation and seccomp in Firecracker documentation.

The following figure shows a rough sequence of events that lead to Fargate agent running customer containers on a VM when starting an ECS task.



Fargate container start sequence

The Fargate agent receives a message that it needs to start a task. This message also contains details about the elastic network interface (ENI) that has been provisioned for the task.

It then sets up the networking for that task by creating a new network namespace and provisioning the network interface to this newly created network namespace.

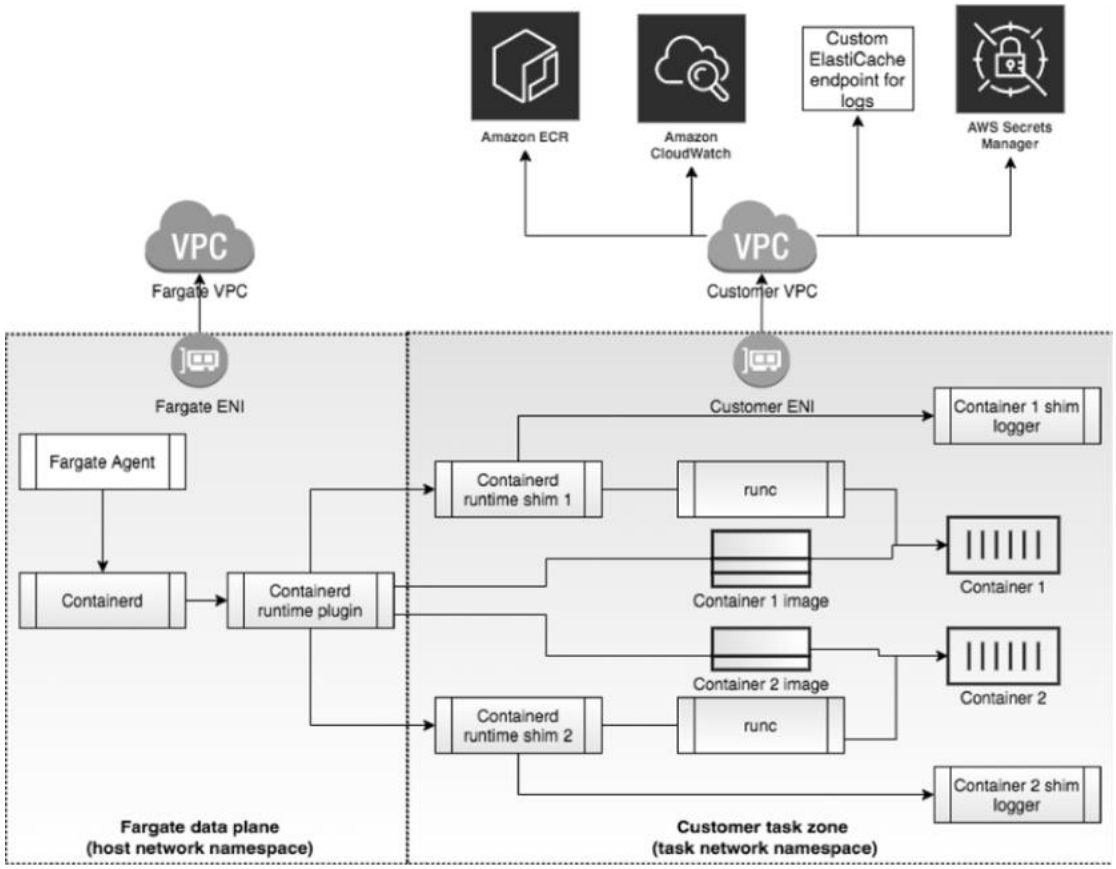
Next, it downloads the container images, any secrets and configuration needed to bootstrap containers using the ENI from customer's account.

Customer containers are started next using Containerd APIs.

Containerd in turn creates shim processes that server as parent processes for containers. These shim processes are also used to spin up containers using runC.

The Fargate agent also specifies what kind of Containerd logger shims need to be started based on the configuration specified by customers. Containerd uses this to start logging plugins for containers.

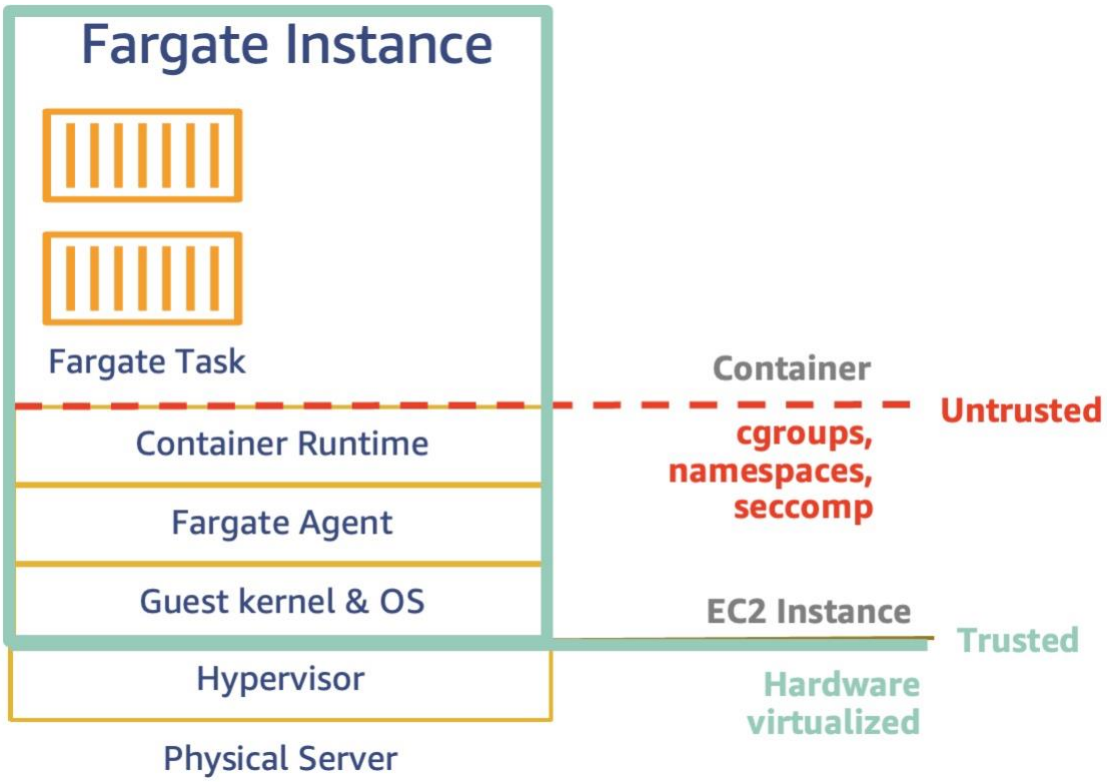
The following diagram shows how a high-level view of how Fargate data plane components are laid out.



Fargate components

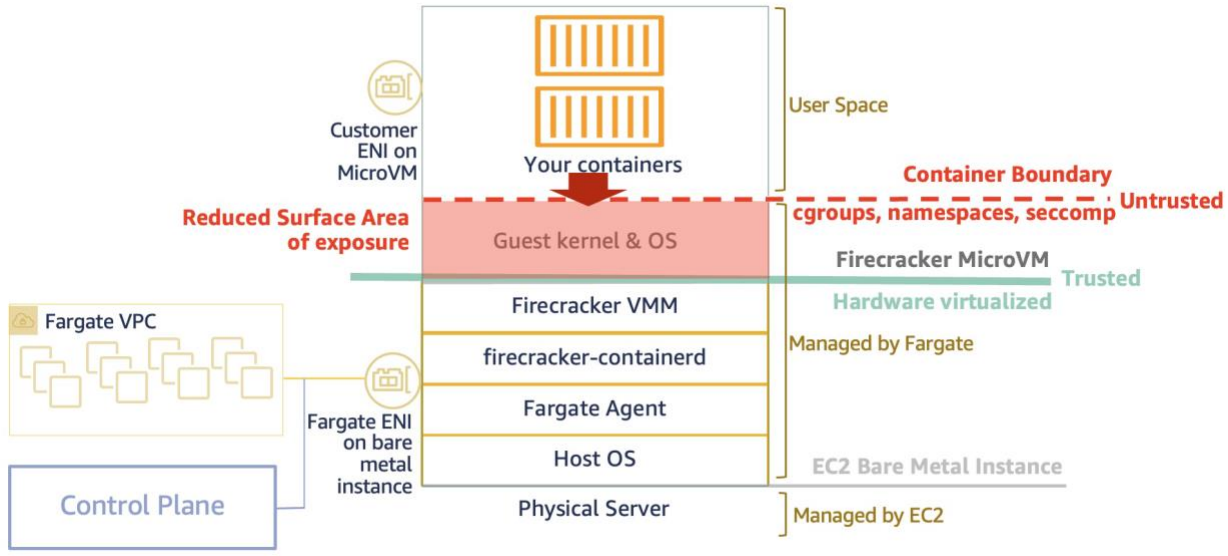
Security in Fargate

Isolation by Design



Fargate data plane components

EC2: At the bottom of the Fargate Data Plane stack, the EC2 hypervisor uses trusted hardware virtualization to isolate instances running on the same physical server. As mentioned in the previous section, the EC2 instance runs Amazon Linux 2, Fargate Agent, and a container runtime. The container isolation boundary is composed of abstractions like cgroups, namespaces, and seccomp policies. Although they provide some isolation, we decided to avoid collocating tasks by running a one-to-one mapping of tasks to instances. This execution model offers multiple layers of isolation. Hence, Fargate never collocates two tasks on the same EC2 instance, even from the same customer. Each instance runs one and only one task. When RunTask API is called, a fresh EC2 instance is used, and when the task stops, the instance is terminated. This provides a task-level isolation guarantee for Fargate on the EC2 data plane.



Fargate isolation with Firecracker

Firecracker: A Firecracker microVM is created for every task placed on the host. All the containers belonging to the task are executed in the microVM to facilitate a secure task boundary. Firecracker is responsible for securing the microVM. The firecracker process itself will be jailed using cgroups and seccomp profiles. Container/task credentials and secrets will be made available to only the appropriate Firecracker microVM. Hence the primary security consideration in Firecracker is to isolate tasks from each other reliably.

Network Security

Security groups block communication between the Fargate Data Plane EC2 instances to segregate instance-to-instance traffic within the Fargate VPC. VPC flow logs, CloudTrail, and DNS logs are enabled and monitored to detect and respond to security events within the Fargate VPC.

For Firecracker, the network interfaces are attached and made available only to the VMs where tasks/pods run. A dedicated ENI in the customer VPC is created and attached to the customer task/pod.

Security groups for pods integrate Amazon EC2 security groups with Kubernetes pods in Amazon EKS. You can use Amazon EC2 security groups to define rules that allow inbound and outbound network traffic to and from pods that you deploy to nodes running on many Amazon EC2 instance types and Fargate. See Introducing security

groups for pods blog post and related documentation for a detailed explanation of this capability.

Patching and security updates

Fargate deploys security updates and patches automatically based on the platform version. If a security issue is found that impacts a platform version revision, AWS creates an updated platform version revision. The platform version revisions are immutable; when Fargate needs to address a CVE, Fargate creates a new platform version revision and deploy that across the fleet of instances, followed by retiring old tasks.

The standard patching process for ECS on Fargate has three milestones. First, a patched platform version is released. This milestone comprises evaluating and preparing the patch for the platform version, building the AMI with patched packages, scale and regression testing, and finally, deployment to production. At this point, all new task launches happen on fully patched platform versions. Next, impacted customers are sent a task retirement notification. For non-managed tasks that are started with the RunTask API, after the patched platform version is deployed to production, a Personal Health Dashboard (PHD) notification is sent to inform customers that have running tasks impacted by the CVE. Usually, customers are given 15 calendar days to relaunch their impacted tasks. The ECS service scheduler automatically replaces tasks that run as part of an ECS service. Finally, after the deadline specified in PHD notification, impacted customer tasks still running are retired by Fargate task retirement workflows. The scheduler automatically restarts managed tasks on the patched platform version. Unmanaged tasks are stopped by workflow and require customer intervention to be restarted. For more information, see Task retirement.

EKS on Fargate also has a similar three-step process for patching. First, a patched task image is released by the EKS Fargate team. This milestone involves evaluating and building the necessary changes into the task image and running the EKS Fargate regression test suite. After testing, this new task image is deployed so that pods within new Fargate Profiles use this new patched task image. Second, all existing EKS Fargate Profiles are updated so that any new pods created within these existing Fargate Profiles use the new patched task image as well. Third, depending on the severity, EKS Fargate may terminate existing pod deployments so that pods are relaunched with the new patched task image. When EKS Fargate has to terminate existing pods, we leverage the Kubernetes eviction API so that any Pod Disruption Budgets(PDBs) configured, which is highly recommended, are respected. If EKS Fargate is unable to gracefully evict a pod due to overly restrictive PDBs, we create an Event Bridge event

notifying the customer that we will forcibly evict the pod at a specific time in the future, by default in 5 days. Customers have until then to update their PDBs to allow graceful eviction of the pod so that it can be terminated and relaunched with the new patched task image. If customers do not update their PDBs to allow for graceful eviction before the specified time, EKS Fargate will forcibly terminate the pod at the time specified which has the potential to be disruptive.

Since patching and security updates are integral parts of achieving compliance (e.g., PCI-DSS requirement 6.2), please refer to the Fargate Compliance section in this document to further understand security controls in place for achieving the third-party security certifications and attestations for Fargate.

Storage Security

Fargate supports the two types of storage:

- Ephemeral storage
- Amazon EFS volumes for persistent storage

With platform version 1.4.0, Fargate offers a single 20GiB ephemeral storage volume. Amazon ECS tasks launched in PV 1.4 benefit from server-side encryption of the 20GiB ephemeral storage using Fargate-managed keys. Amazon EKS pods launched in Fargate also use this feature as Amazon EKS uses Fargate PV 1.4.

EKS pods running on Fargate have slightly less usable storage as about 1 GiB is used by the required Kubernetes components (`kubelet`, `kube-proxy`, and `containerd`).

To use Amazon EFS volumes for persistent storage with Fargate, refer to the three part blog series. For encrypting EFS volumes in general, including data at rest, refer to the Encrypting File Data with Amazon EFS white paper. Refer to this walkthrough for enforcing encryption on an Amazon EFS file system at rest.

Fargate Compliance

Amazon ECS on Fargate meets the standards for PCI DSS Level 1, ISO 9001, ISO 27001, ISO 27017, ISO 27018, SOC 1, SOC 2, SOC 3, and HIPAA eligibility. We are in the process to achieve similar compliance for EKS on Fargate. Third-party auditors assess the security and compliance of Amazon ECS and Amazon EKS as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others. For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS

Compliance Programs. You can download third-party audit reports using AWS Artifact. For more information, see Downloading reports in AWS Artifact.

HIPAA

Fargate is HIPAA eligible. If you have an executed Business Associate Addendum (BAA) with AWS, you can process encrypted Protected Health Information (PHI) using containers deployed onto Fargate in an account designated as a HIPAA account.

For more information, please visit our page on HIPAA compliance. If you plan to process, store, or transmit PHI and do not have an executed BAA from AWS, please see AWS Compliance for more information.

Processing sensitive Controlled Unclassified Information (CUI)

Fargate is available in AWS GovCloud (US-East) and (US-West) Regions. For a full list of AWS Regions where Fargate is available, please visit our Region table.

Your compliance responsibility when using AWS is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and describe steps for deploying security-hardened baseline environments on AWS.

Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how customers can run sensitive workloads regulated under the U.S. Health Insurance Portability and Accountability Act (HIPAA).

AWS Compliance Resources – This collection of workbooks and guides that might be applicable to your industry and geography.

AWS Config – This AWS service enables you to assess, audit, and evaluate the configurations of your AWS resources. Config helps you monitor configuration data for container-based resources, such as monitoring configuration changes to EKS cluster settings and tracking compliance for cluster configurations. AWS Config supports ECS, EKS, and Amazon ECR.

AWS Security Hub – This AWS service helps you assess your high-priority security alerts and security posture across AWS accounts in one comprehensive view. Security Hub has built in mapping capabilities for common frameworks like CIS, PCI DSS, and more.

Fargate Security Partners

The following AWS Partner products are available to help you add additional security and features to Fargate workloads.

Aqua Security

You can use Aqua Security to secure your cloud-native applications from development to production. The Aqua Cloud Native Security Platform integrates with your cloud-native resources and orchestration tools to provide transparent and automated security. It can prevent suspicious activity and attacks in real time, and help to enforce policy and simplify regulatory compliance.

Palo Alto Networks

Palo Alto Networks provides security and protection for your hosts, containers, and serverless infrastructure in the cloud and throughout the development and software lifecycle. Twistlock is supplied by Palo Alto Networks and can be integrated with Amazon ECS FireLens. With it, you have access to high fidelity security logs and incidents that are seamlessly aggregated into several AWS services. These include Amazon CloudWatch, Amazon Athena, and Amazon Kinesis. Twistlock secures workloads that are deployed on AWS container services.

Sysdig

You can use Sysdig to build, detect and respond to threats, continuously validate cloud posture and compliance, and monitor performance. The Sysdig Secure DevOps Platform has embedded security and compliance features to protect your cloud-native workloads, and offers enterprise-grade scalability, performance, and customization. Sysdig has also authored Fargate-specific blogs like Securing Fargate workloads: Meeting File Integrity Monitoring (FIM) requirements, Falco Support on Fargate and Digging into Fargate runtime security approaches: Beyond ptrace and LD_PRELOAD.

Conclusion

This document explains how AWS Fargate enables you to securely run containerized workloads without managing servers. This whitepaper described the key benefits of

Fargate, its suitability for different types of applications, and its security model. It also reviews the security measures used in AWS Fargate infrastructure to protect customer data and workloads. For more information about Fargate, please visit AWS Fargate webpage.

Contributors

- Re Alvarez Parmar, Principal Specialist Solutions Architect, AWS
- Paavan Mistry, Senior Developer Advocate, AWS

Further Reading

For additional information, refer to:

- Amazon EKS Best Practices Guide
- Amazon ECS Best Practices Guide
- Shared Responsibility Model
- AWS Security Best Practices in IAM
- Under the hood: AWS Fargate data plane
- The open-source AWS for Fluent Bit project on GitHub
- AWS Architecture Center

Document revisions

Date	Description
April 15, 2022	First publication