aws modern apps

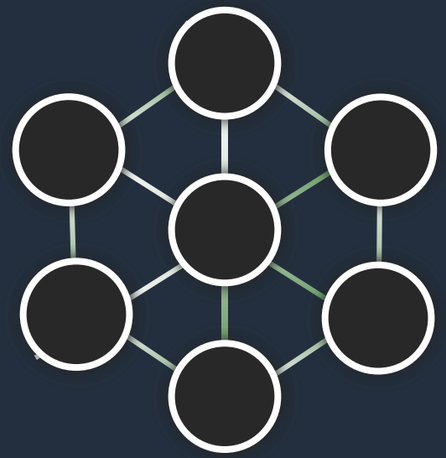
Architecting Secure Serverless and Containerized Applications

Jeremy Cowan
Principal Developer Advocate

Josh Kahn
Principal Solutions Architect

Microservices emerge as companies grow…

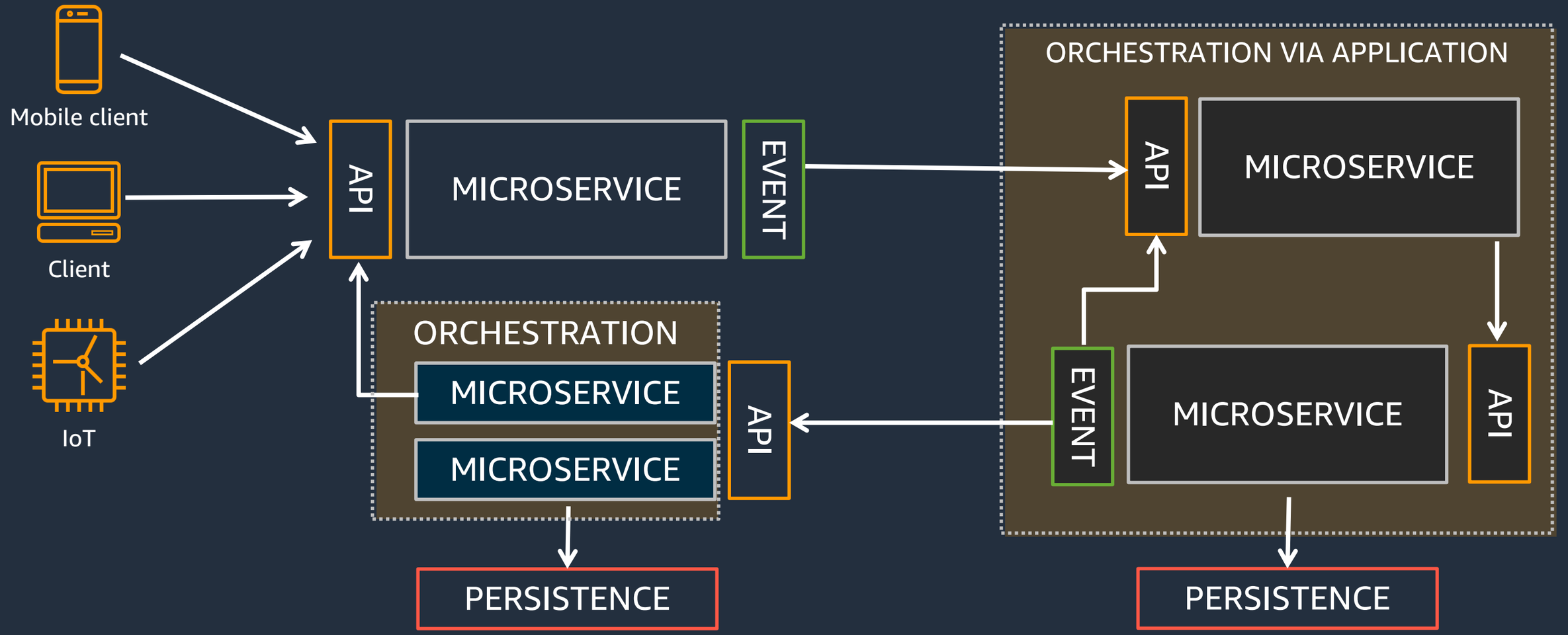
Monolith
Does everything



Microservices
Does one thing

aws modern apps

Microservices manifest throughout workload



aws modern apps

(Subset of) Options to build microservices

AWS Lambda

AWS Fargate

Amazon ECS

Amazon EKS

aws modern apps

Comparison of operational responsibility

	AWS manages	Customer manages
Lambda Serverless functions	• Data source integrations • Physical hardware, software, networking, and facilities • Provisioning	• Application code
Fargate Serverless containers	• Container orchestration, provisioning • Cluster scaling • Physical hardware, host OS/kernel, networking, and facilities	• Application code • Data source integrations • Security config and updates, network config, management tasks
Amazon ECS/ Amazon EKS Container management as a service	• Container orchestration control plane • Physical hardware software, networking, and facilities	• Application code • Data source integrations • Work clusters • Security config and updates, network config, firewall, management tasks
Amazon EC2 Infrastructure as a service	• Physical hardware software, networking, and facilities	• Application code • Data source integrations • Scaling • Security config and updates, network config, management tasks • Provisioning, managing scaling and patching of servers

aws modern apps

Security considerations for microservices

- More transient and dynamic
- More distributed and complex
  - More services interdependencies over network
  - Scheduling / scaling / resource management
- Isolation is similar to virtual machines, but different:
  - May share a kernel
  - May share a network and a network interface

Security is Everyone's Responsibility

aws modern apps

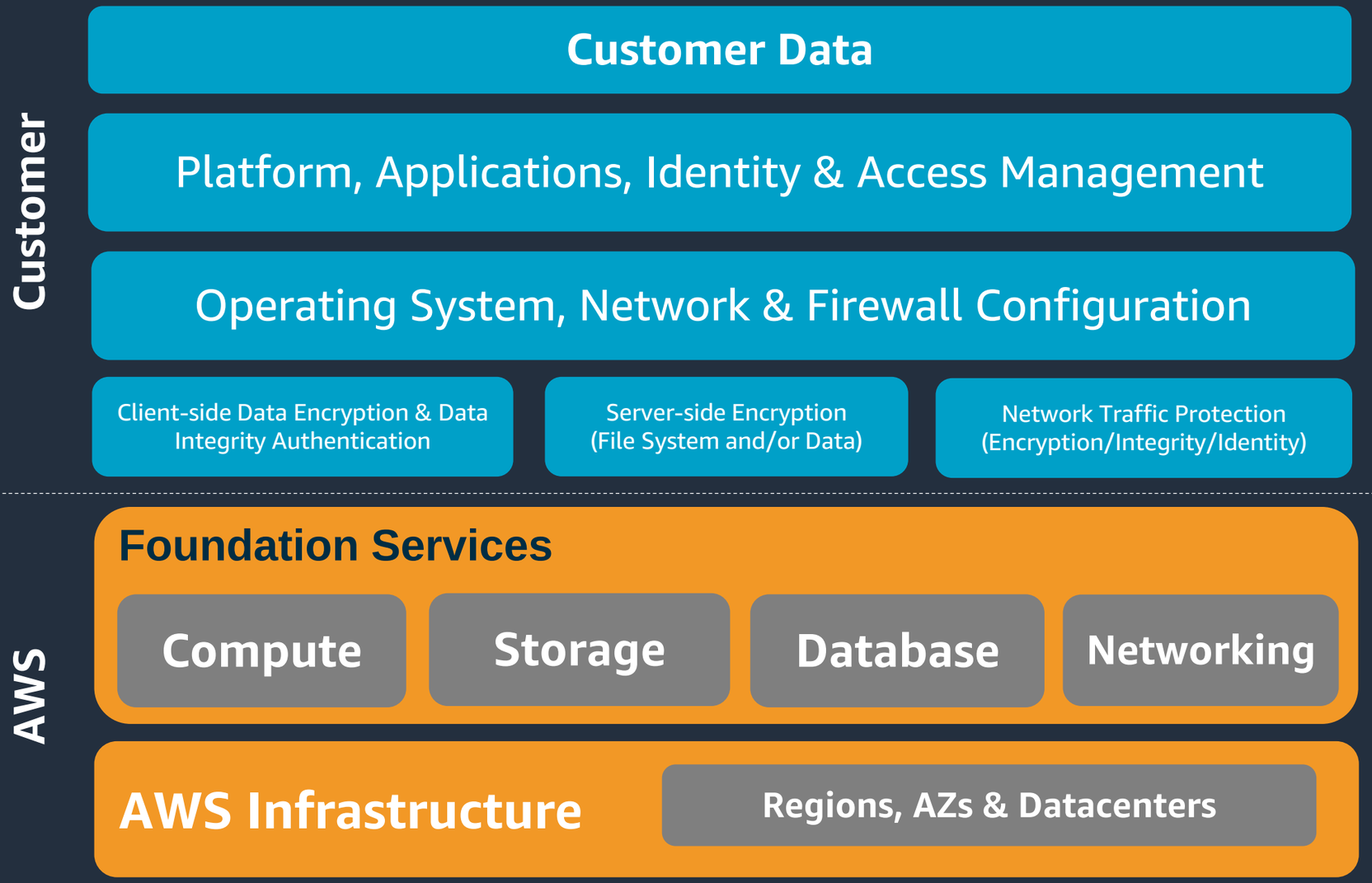
Four principles of securing modern applications

1. Shared responsibility model
2. Least privilege
3. Defense in depth
4. Secure your software supply chain

Security Principle #1: Shared Responsibility with AWS

aws modern apps

Security principle #1: Shared Responsibility with AWS

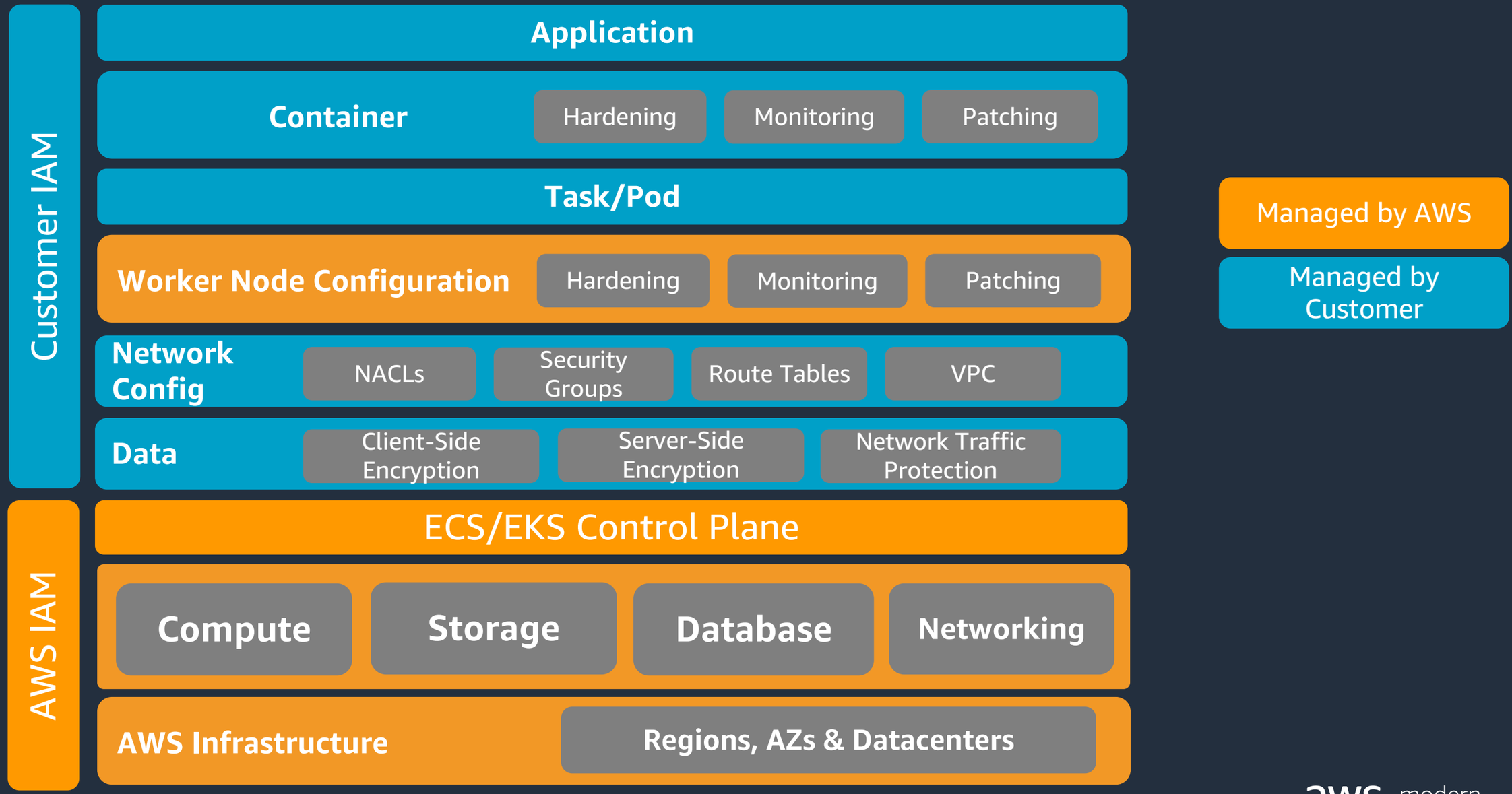


Customers are responsible for their security and compliance **IN** the Cloud

AWS is responsible for the security **OF** the Cloud

aws modern apps

Responsibilities change with Amazon on Fargate



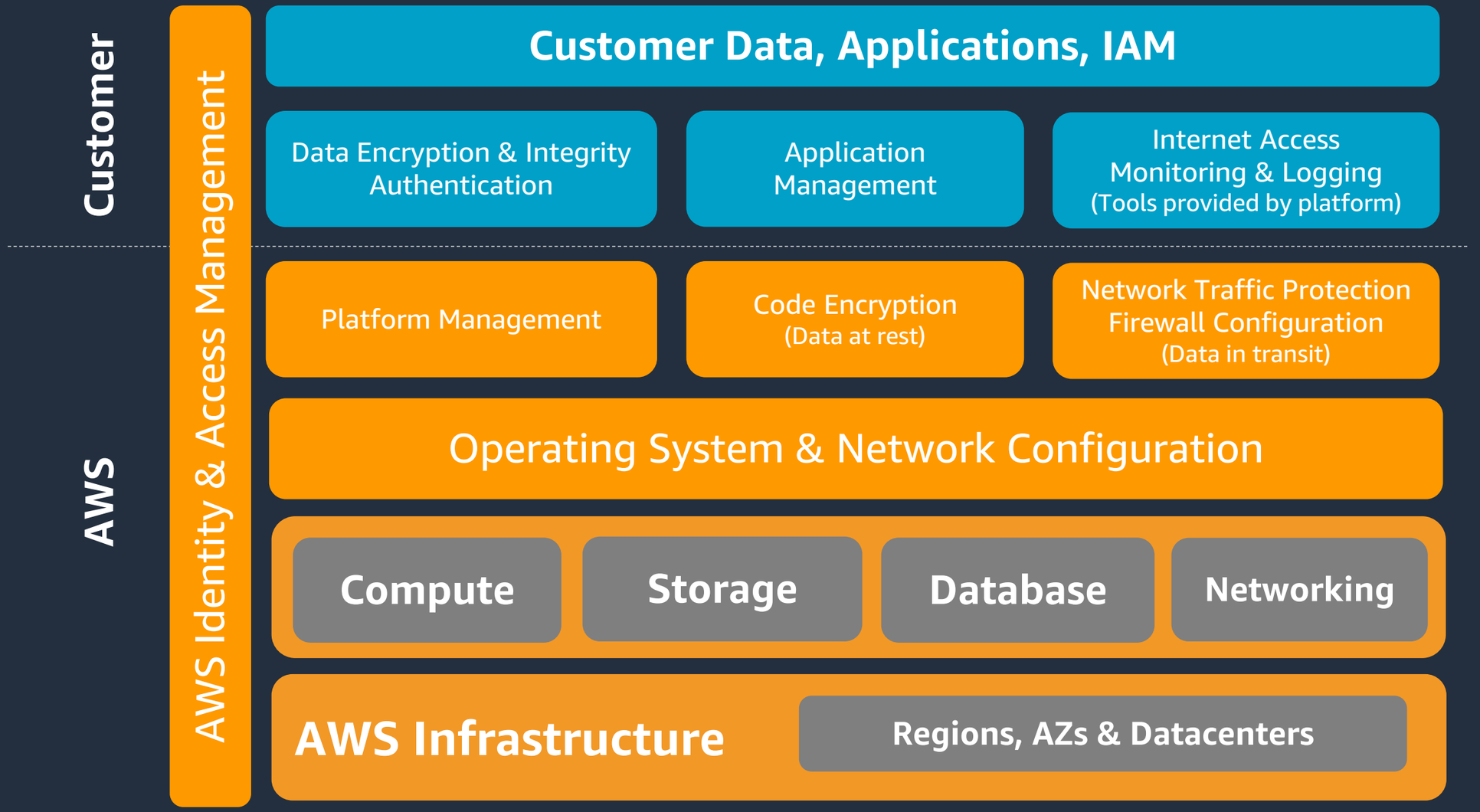
aws modern apps

Security benefits of AWS Fargate

We do more, you do less.

- Patching (OS, Docker, Amazon ECS agent, etc.)
- Task isolation
- No --privileged mode for containers
- AES-256 Server side encryption of ephemeral storage

With Serverless, AWS takes a greater share of responsibility



aws modern apps

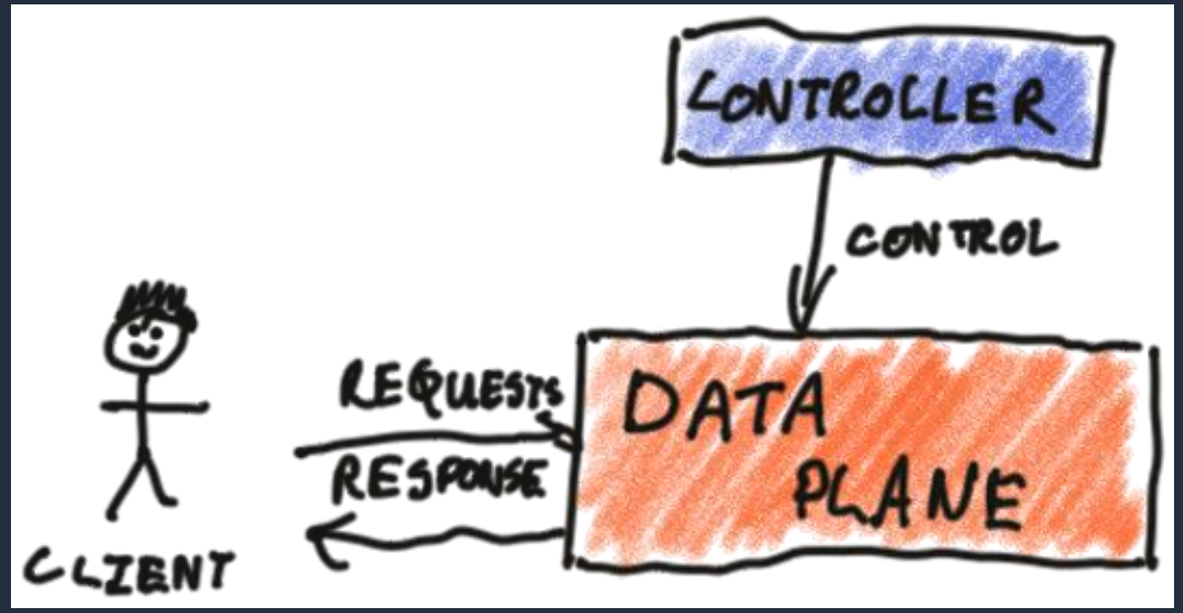
Lambda service composed of control plane and data plane

Control Plane

- Management APIs, such as:
  - `CreateFunction`
  - `UpdateFunctionCode`
- Requires IAM permission to access

Data Plane

- Invoke Lambda function via
  - `Invoke`
- Requires IAM permission to access
- When invoked, data plane runs code on:
  - Existing execution environment, if exists
  - New environment, after allocation



@MarcJBrooker

aws modern apps

Container orchestration

AWS managed control planes

- Elastic Container Service (ECS)
- Elastic Kubernetes Services (EKS)

Responsible for managing the scheduling and lifecycle of containers

Data plane

- Self managed EC2
- Managed node groups (EKS only)
- Fargate (ECS and EKS)

Security Principle #2: Least Privilege

aws modern apps

Security principle #2: Least Privilege

- Grant only the essential privileges needed to perform intended work
- Attach to compute via execution role
  - Prefer unique role per function or task
  - Enforce permission boundaries
- Be specific: identify limited set of resources and actions allowed
  - Scrutinize use of “*”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "xray:PutTraceSegments",
                "xray:PutTelemetryRecords"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::my-bucket",
                "arn:aws:s3:::my-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```


aws modern apps

Use AWS IAM to assign and audit fine-grained permissions

- IAM roles can be assigned to:
  - ECS Tasks
  - Kubernetes Pods
  - Lambda Functions
  - Users
- Allow (or deny) access to AWS APIs (management, data planes)
- Periodically audit access
  - AWS Access Advisor
  - Amazon CloudTrail Insights
  - Kubernetes audit log/CloudWatch

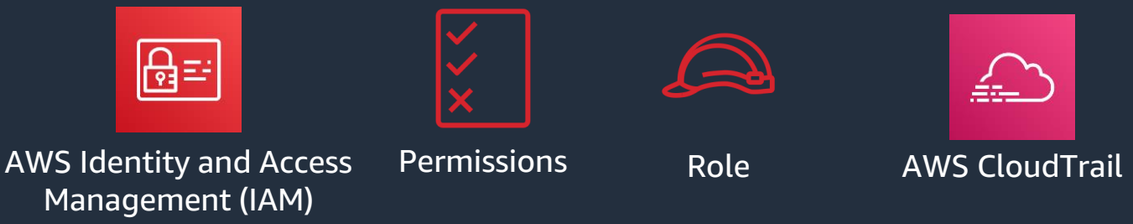
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::my-bucket",
                "arn:aws:s3:::my-bucket/*"
            ],
            "Effect": "Allow",
          "Conditions": {
            "StringEquals": {
              "aws:PrincipalOrgId": "o-xxxxxxxxxx"
            }
          }
        }
    ]
}
```


aws modern apps

User access

Apply principles of least privilege.

- Authenticate all user access to hosts and containers.
- Implement IAM policies and roles to restrict access to only required services.
- Restrict access and write permissions to image registry.



Security, Governance, and Oversight

Authentication
+
Authorization
+
Audit/Log

aws modern apps

Security Principle #3: Practice Defense in Depth

aws modern apps

Common vectors of attack

App Vulnerabilities

- SQL Injection
- Cross-site Scripting (XSS)
- OWASP Top 10
- Common Vulnerabilities and Exposures (CVE)

Dependencies

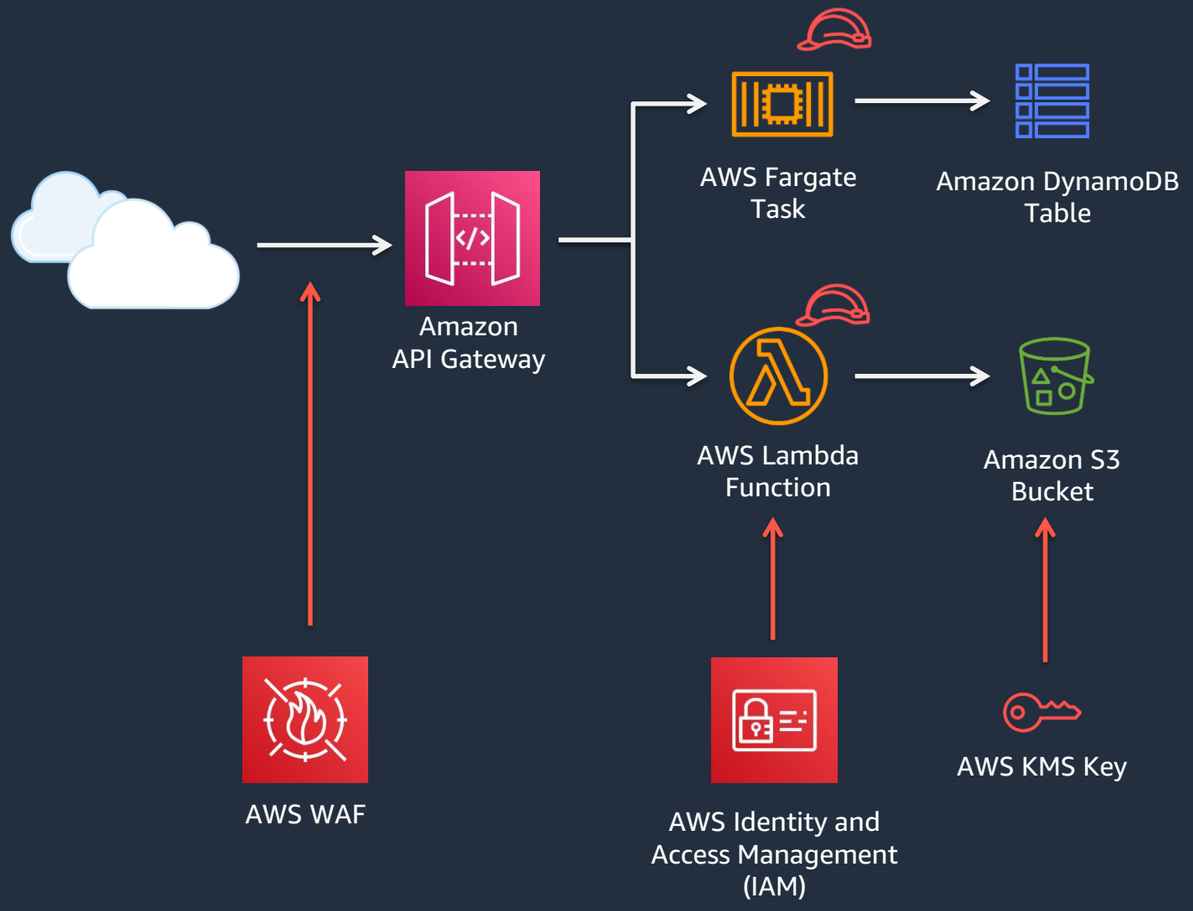
- Libraries
- Distributions
- Base Images

Host / Network

- Patching
- Network Segmentation

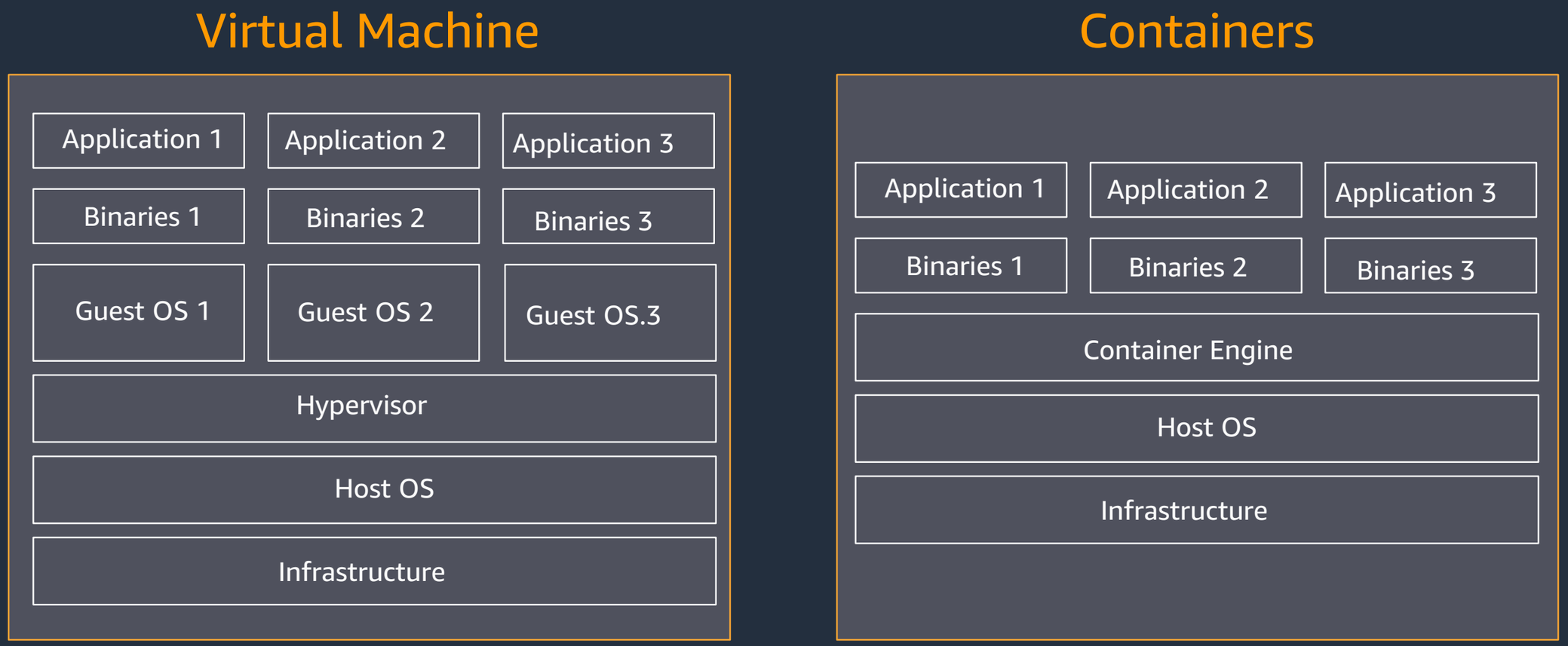
Security principle #3: Practice Defense in Depth

- Implement multiple, redundant measures across system to address common attack vectors
- Leverage AWS managed services and integrations
- Consider service features, e.g. backup and encryption

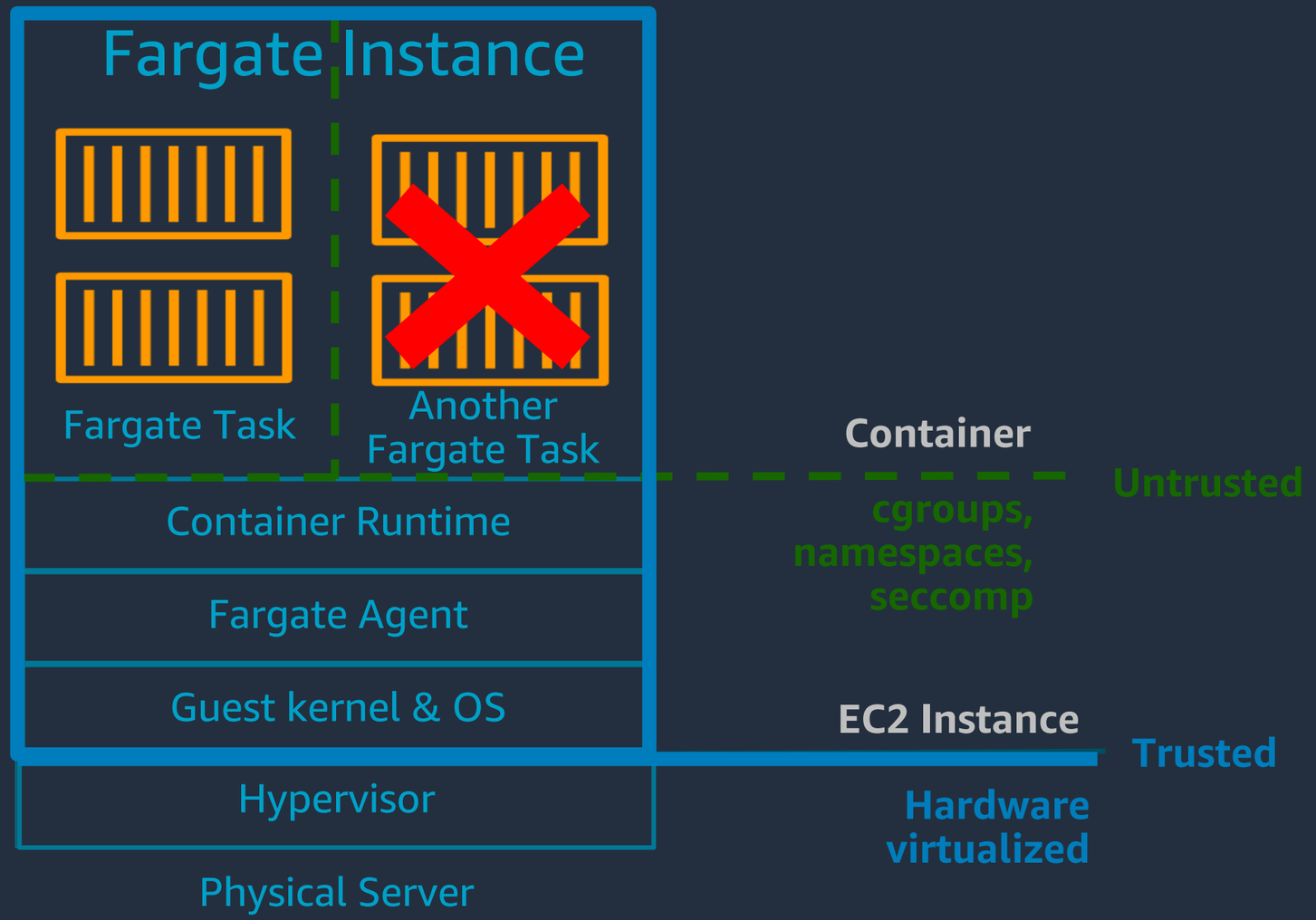


aws modern apps

Container versus Virtual Machine



aws modern apps



One & only one task per EC2 instance

aws modern apps

Containers: Runtime security options

- Containers run as processes on the Linux kernel
- Linux options:
  - cgroups
  - namespaces
  - Linux capabilities
  - seccomp*
  - AppArmor*
  - SELinux*

- 3rd party and open source security options include:
  - Aqua
  - Falco (CNCF project)
  - PA Primsa
  - Redhat StackRox
  - Sysdig Secure

* Not applicable to serverless containers (Fargate)

Containers: Network security options

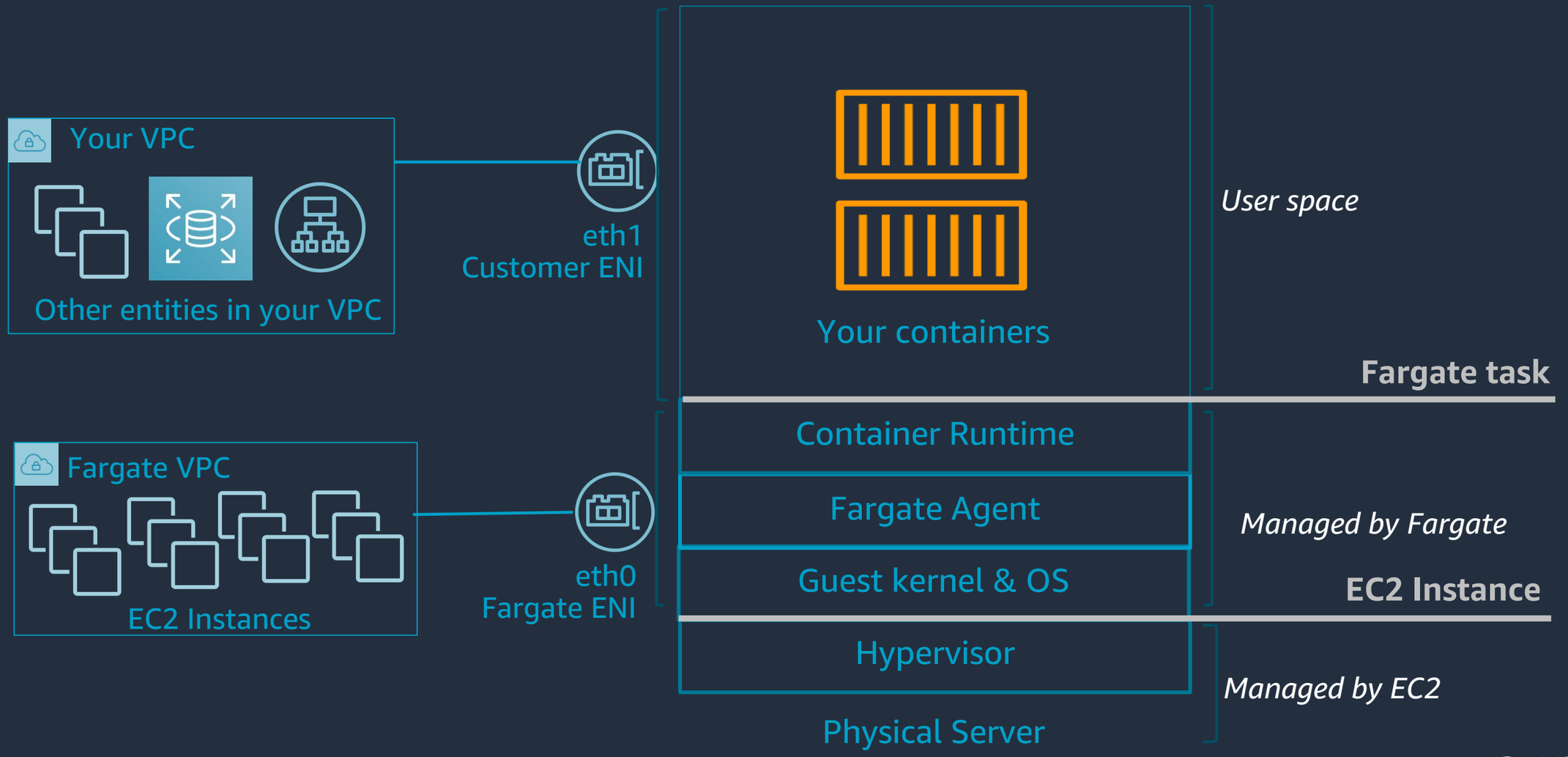
- Restrict communication between:
  - Pods and Tasks
  - Containerized applications and other resources that run within or outside the VPC
- Encrypt traffic between:
  - Pods, Tasks, Instances, Lambda functions (future)
  - AWS load balancers and tasks/pods

Service specific options:

- EKS
  - Kubernetes Network Policies
  - Security Groups for Pods
  - App Mesh (TLS & mTLS)
  - SSL/TLS (load balancing/ingress)
- ECS
  - Security Groups for Tasks
  - App Mesh (TLS & mTLS)
  - SSL/TLS (load balancing)

aws modern apps

Fargate networking: A deeper look



aws modern apps

Fargate: Process isolation

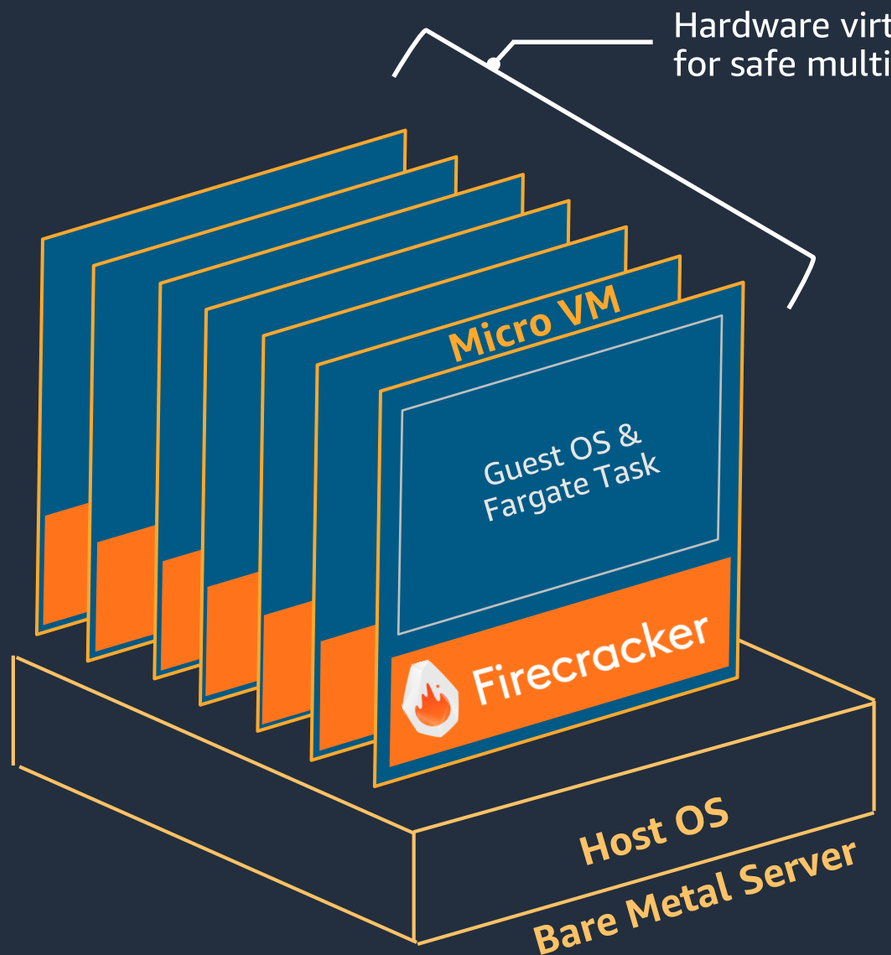
Fargate implements a shared nothing architecture

- Disk
- Memory
- CPU
- Network

Each task/pod runs as a separate virtual machine (EC2 or Firecracker)

Both VM types provide a hard security boundary

Firecracker

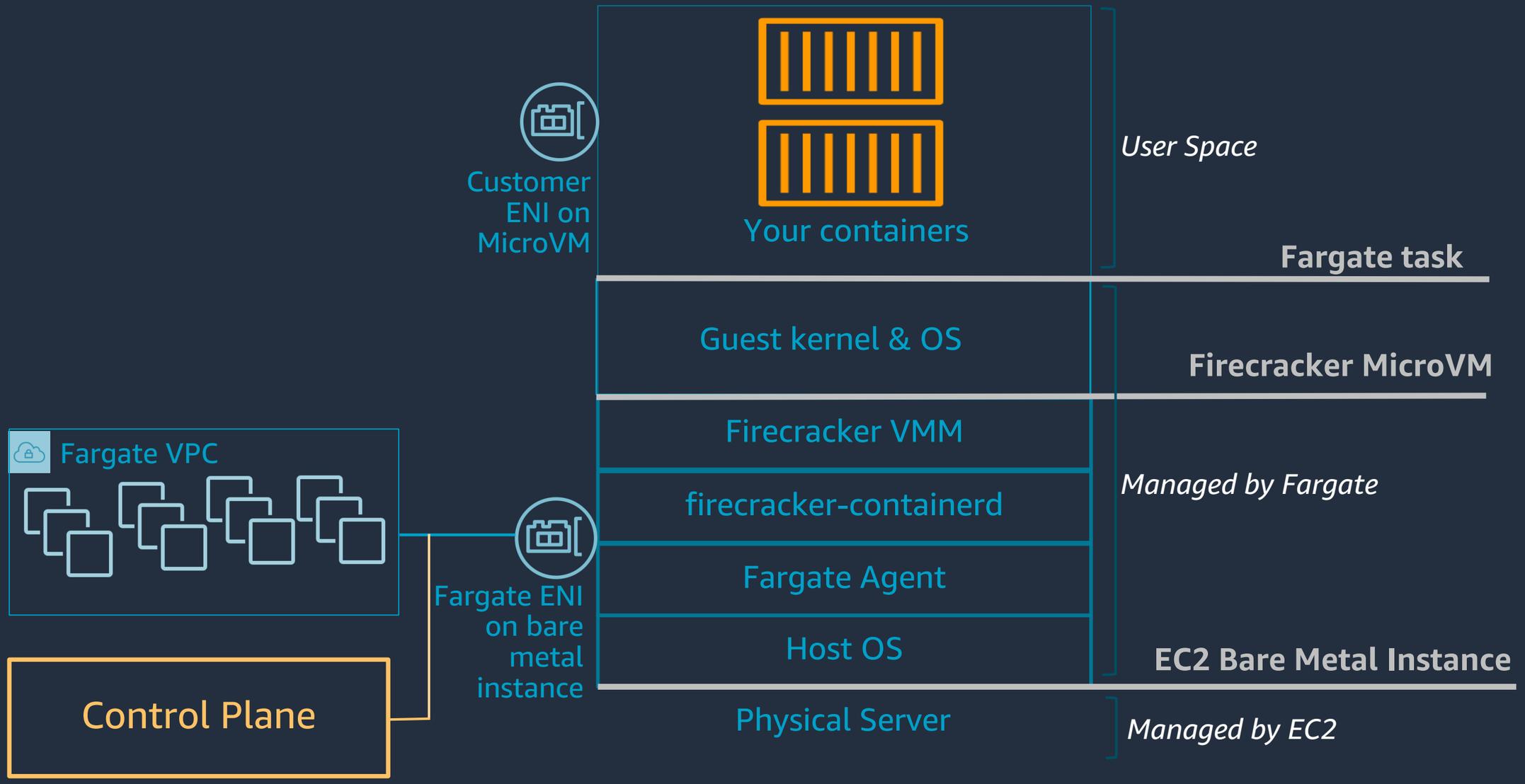


Firecracker is built on KVM, the same hypervisor that EC2 Nitro instances are built on.

Hardware virtualization ensures that tasks from different customers can run safely on the same physical machine.

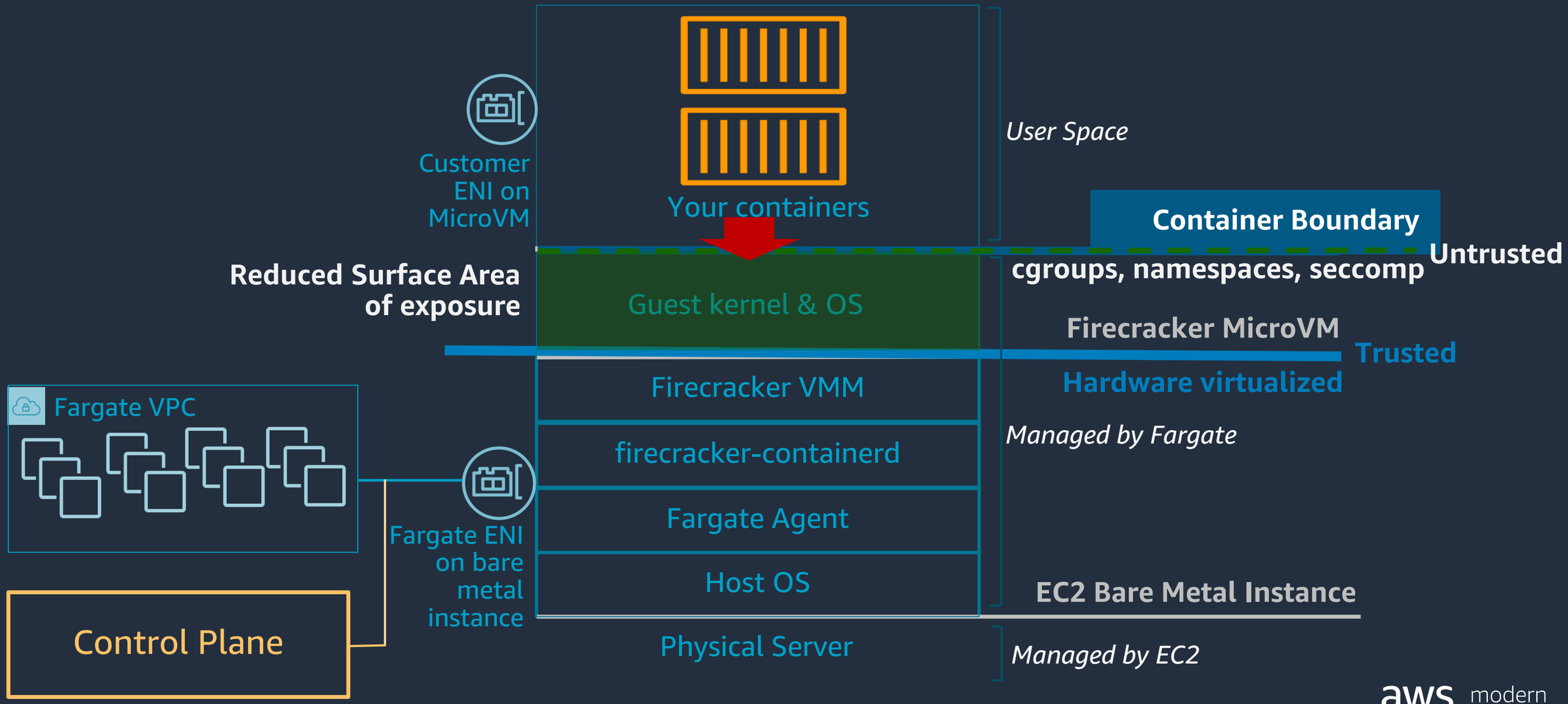
aws modern apps

Fargate on Firecracker networking: A deeper look



aws modern apps

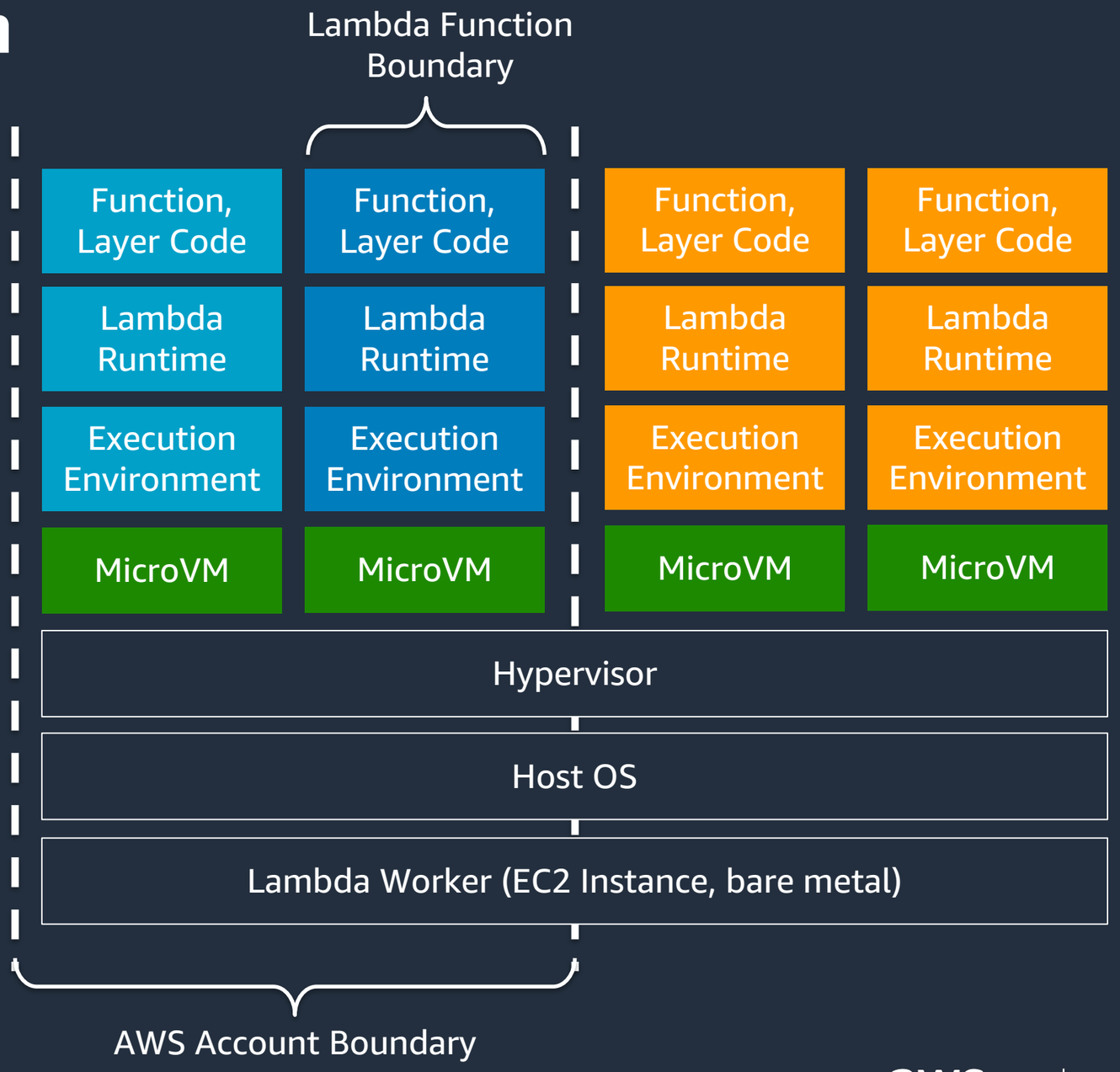
Firecracker enhances isolation of tasks



aws modern apps

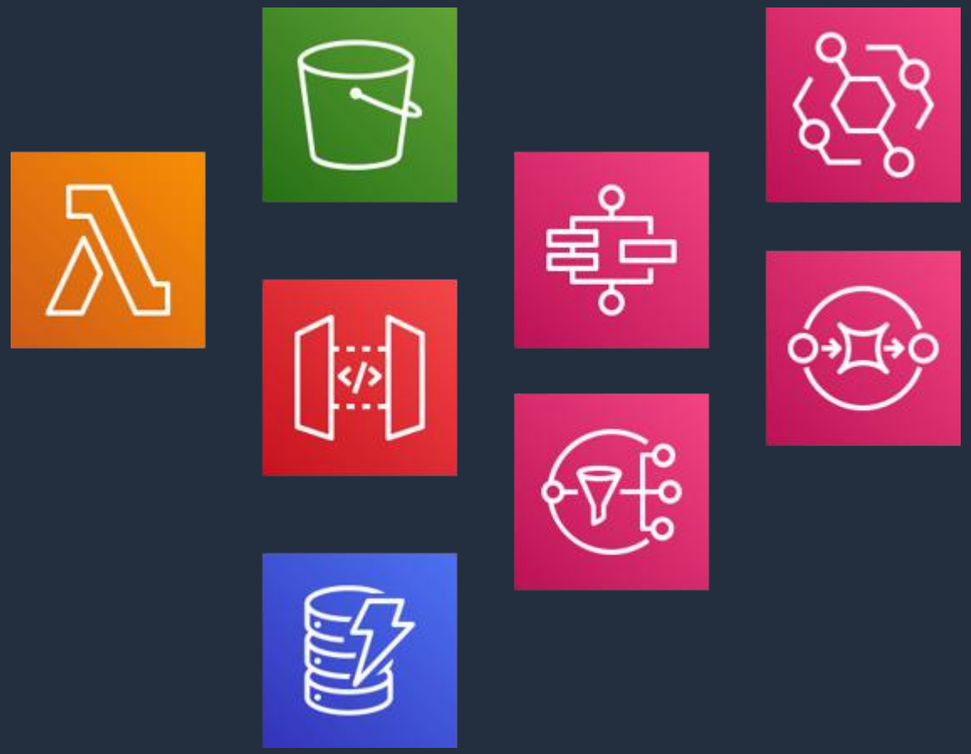
Lambda Function isolation

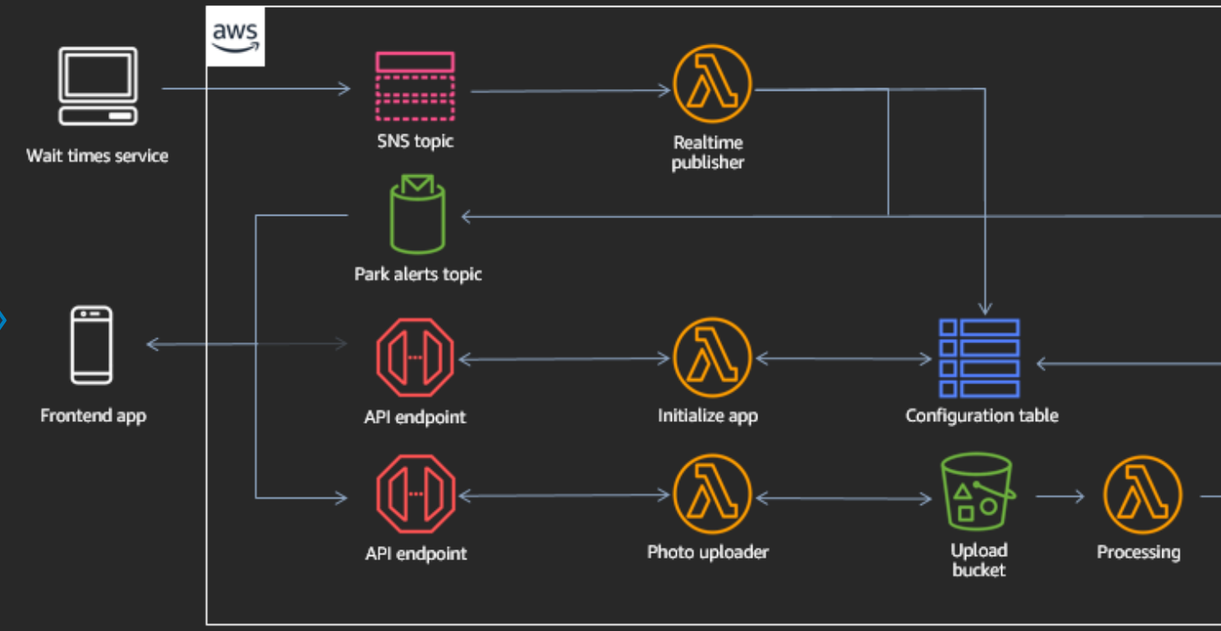
- Each function runs in a dedicated execution environment
  - Each execution environment handles one concurrent invocation
- Execution environment may be reused between invocations
  - Use caution when storing sensitive data in memory or `/tmp`
- AWS maintains runtime and execution environment
  - Patching, etc.
  - Does not apply to container packaging



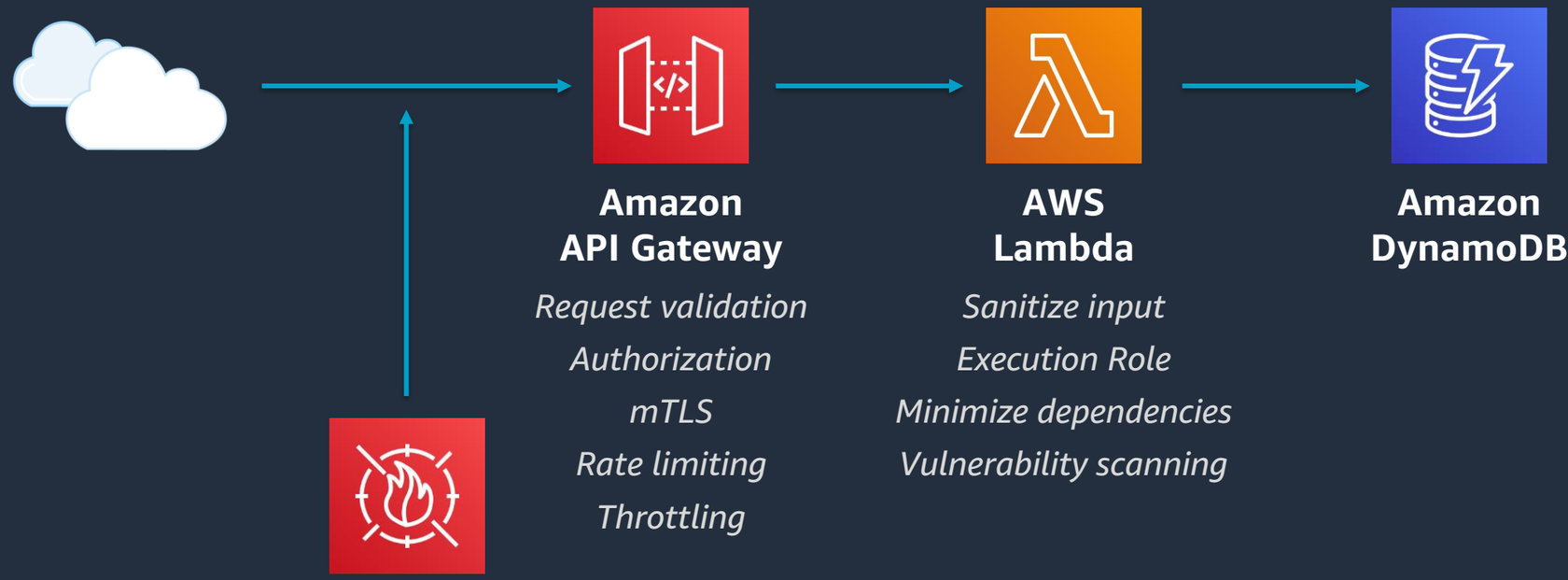
aws modern apps

Serverless architectures are small pieces, loosely joined





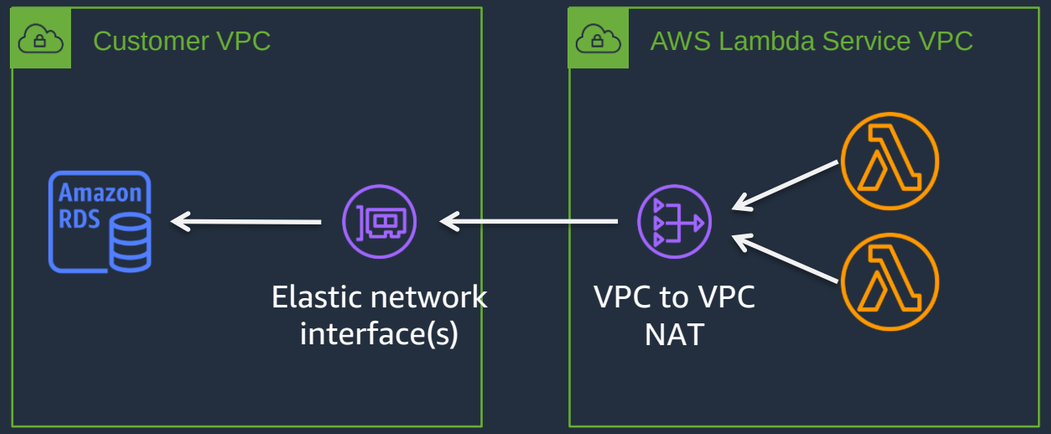
Securing a Serverless web service



aws modern apps

Common ask: Should my Lambda function be VPC-enabled?

- Lambda functions always run in VPCs owned by the Lambda service team
  - When VPC enabled, configured with access to your VPC via an ENI
- Lambda functions are always invoked via `Invoke` action
  - Access controlled by AWS IAM
- Answer: Only if your function:
  - Needs access to resources in the VPC
  - Desire to restrict outbound network path



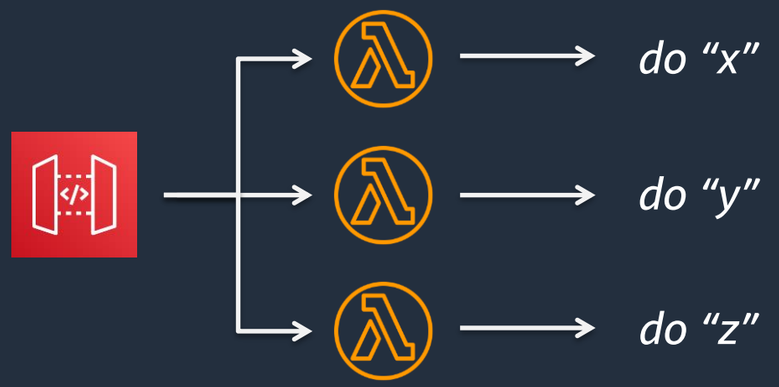
aws modern apps

Security Principle #4: Secure Your Software Supply Chain

aws modern apps

Security principle #4: Secure Your Software Supply Chain

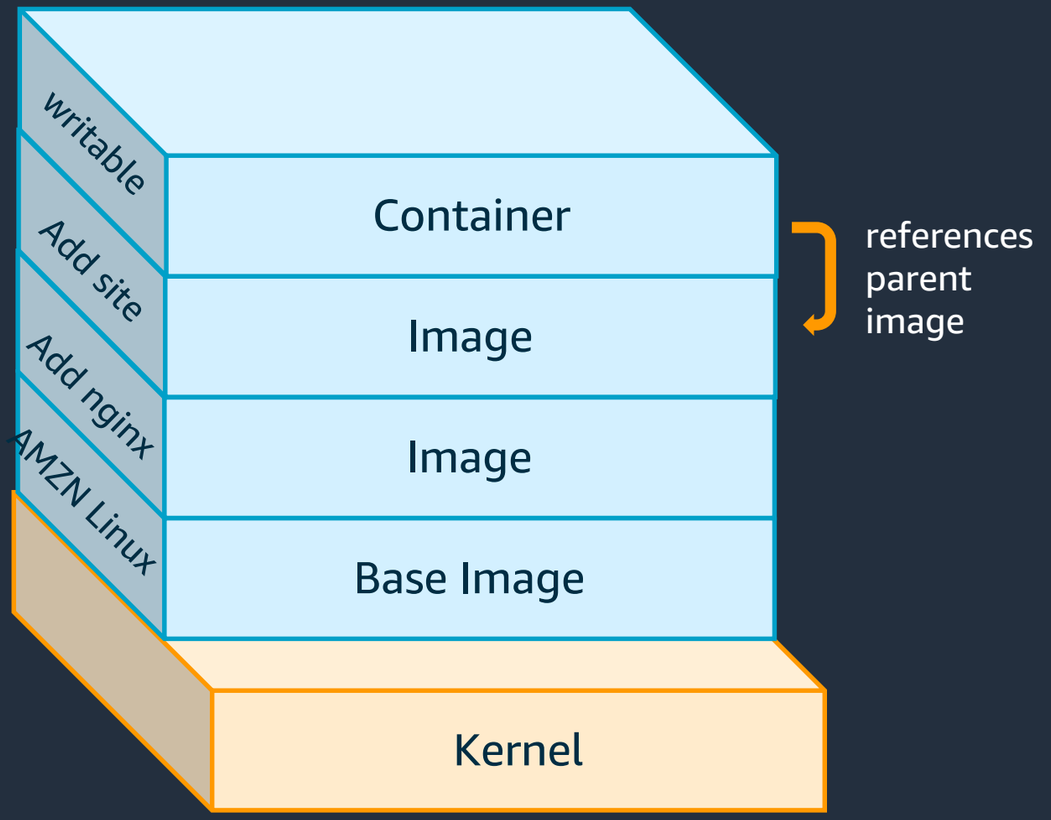
- Keep it simple
  - Prefer single responsibility
  - Easier to debug; cleaner IAM privileges
- Never hardcode secrets in code
  - Use AWS Secrets Manager, Parameter Store
  - Again, never…
- Leverage code and vulnerability scanning
  - Don’t forget dependencies



aws modern apps

Components of the software supply chain

- Base image*
- Language runtime*
- Open source, third-party packages
- Your code



* May be supplied and/or managed by AWS

aws modern apps

Managing dependencies is key

- Understand you dependencies: https://deps.dev/
- Minimize dependencies
- Keep dependencies up-to-date to reduce risk and effort
- Leverage dependency check tools, such as:
  - OWASP
  - Protego
  - Snyk
  - Twistlock
  - Puresec

Build secure container images for Fargate and Lambda

Minimizing the attack surface

- Create images from Scratch
- Create minimal images (docker-slim)
- Use distro-less images without package manager or shell
- Run the application as a non-root user
- “Defang” your containers
- Lint your Dockerfiles with Dockle or Hadolint
- Scan your images for vulnerabilities (CVEs)

Securing your code

Educate about writing secure code

Perform static code analysis (whitebox testing)

Perform dynamic security testing

- Proactively inject faults into the application
- Fuzz testing

Incident Response

aws modern apps

Advanced persistent threats

- Crime syndicates
- State-sponsored
- Script kiddies
- Hacktivists, e.g. Anonymous

- Will exploit weaknesses in your defenses
- Will use social engineering
- May be employees
- Attacks are increasing in frequency and sophistication

Have a plan

- Don’t panic
- Create an incident response plan
  - Prevention
  - Collection
  - Remediation
- Practice your response to an incident

- Avoid becoming the next Colonial Pipeline
  - Help Wanted: Develops, validates, and maintains an incident response plan and processes to address potential threats.