# **Cloud Deployment Artifacts and Workload Optimization**

We distinguish technologies that provide environments where applications can execute and serve the purpose of running code as deployment options. All options involve the allocation and management of computing resources (CPU, memory, storage) and require network connectivity to interact with other systems and the internet. Different deployment options offer varying levels of abstraction, resource allocation, and management complexity. Understanding their characteristics helps in choosing the right approach for specific use cases. We distinguish the following models

**Serverless functions** are event-driven, stateless, and scale automatically but are ephemeral.

**Containers** provide lightweight process isolation and share the OS kernel.

**Orchestrated containers** use tools like Kubernetes to manage clusters of containers dynamically.

**VMs** offer full OS virtualization, leading to stronger isolation but higher overhead.

**Mini-VMs** (e.g., Firecracker, Kata Containers) balance security with performance, having lower overhead than full VMs.

**Bare Metal servers** provide maximum control and performance but lack the flexibility of virtualization.

| **Aspect** | **Serverless Functions** | **Containers** | **Orchestrated Containers** | **Virtual Machines (VMs)** | **Mini-VMs (e.g., Firecracker)** | **Baremetal Server** |
| --- | --- | --- | --- | --- | --- | --- |
| **Abstraction**  **Level** | Function-level (high) | Process-level | Multi-container application | OS-level | Lightweight VM | Hardware-level |
| **Resource Allocation** | Fully managed, ephemeral | Shared kernel | Dynamic allocation via orchestrator | Dedicated resources per VM | Lower overhead per VM | Full machine control |
| **Startup Time** | Milliseconds | Seconds | Seconds | Minutes | Milliseconds | Instant (if running) |
| **Performance** | High but cold start penalty | High | High | Moderate to high | High | Highest |
| **Isolation** | Low (multi-tenant) | Moderate (namespaces) | Moderate to high (policy-based) | Strong (hardware-based) | Strong (lightweight VM isolation) | None (full access) |
| **State Management** | Stateless | Usually stateless | Managed via volumes, services | Managed with storage | Managed with storage | Full control |
| **Scaling** | Automatic | Manual or scripted | Automatic via orchestrator | Manual | Fast auto-scaling possible | Manual |
| **Management Overhead** | Lowest (fully managed) | Moderate (manual setup) | High (orchestrator complexity) | High (OS updates, networking) | Moderate (simpler than full VMs) | Highest (full admin) |
| **Cost Model** | Pay-per-use | Resource-based pricing | Resource-based pricing | Pay for VM uptime | Pay per lightweight VM | Fixed cost (hardware) |

**Mapping Deployment Aspects to PCAPNG-Inferred Signals (with Examples)**

| Aspect | What to Look for in PCAPNG | Why It Matters | Example Patterns |
| --- | --- | --- | --- |
| Abstraction Level | Graph structure (flat, tiered, dense); DNS frequency; TTL variance | Serverless = edge bursty traffic; Containers = dense microservice mesh; VMs = flatter structures | Serverless: few-to-one bursty edges; Containers: dense internal cluster edges; VMs: clear tiers, fewer hops |
| Resource Allocation | IP/MAC/port churn; connection duration; NAT/PAT reuse | Ephemeral = short flows & dynamic reuse; Dedicated = stable IPs and long-lived flows | Containers: same MAC used with multiple IPs; Serverless: client IPs vary with short TCP lifetimes |
| Startup Time | Cold start latency; delayed response after idle; flow initiation delay | Millisecond-scale delays after idle suggest serverless cold starts | First packet arrives, no response for ~300–700ms → then function responds |
| Performance | RTT, jitter, retransmits, throughput | Higher variance = lower isolation or orchestration overhead | Orchestrated containers: occasional spikes in RTT due to shared network stack |
| Isolation | Shared MACs/IPs, broadcast storms, ARP/NAT/PAT activity | Shared infra = less isolation; Baremetal/VMs show clean separation | Containers: multiple nodes sharing same IP block; Baremetal: no NAT, no address overlap |
| State Management | Stateless bursts vs. long sessions with storage/DB traffic | Serverless = stateless; Containers/VMs = persistent service-to-storage connections | Serverless: GET calls without session reuse; VMs: sessions followed by DB/storage flows |
| Scaling | Sudden bursts in flow creation/removal; many connections from/to one node | Serverless and K8s scale dynamically; bursts visible as new flows rapidly appear | Orchestrated containers: pod IPs rapidly change; Serverless: API spikes with new client IPs |
| Management Overhead | Control-plane traffic (kube-apiserver, etcd, SSH, SNMP, Ansible, etc.); service discovery protocols | Baremetal/VMs = admin traffic; Containers = orchestrator protocols (K8s, service mesh) | VMs: consistent SSH/Ansible config flows; K8s: etcd, kube-proxy, or CoreDNS traffic |
| Cost Model | Flow duration, idle-to-active ratio, burst frequency | Not directly in PCAP; inferred from session patterns — short-lived = usage-based billing (serverless) | Serverless: 100s of short flows with idle gaps; VMs: long uninterrupted flows all day |

## **Step-by-Step Mapping: From PCAPNG → Role & Community → Deployment Artifact**

FIGURE OUT

### **Connect This to Table:**

| **Deployment Aspect** | **Derived From Analysis** |
| --- | --- |
| **Abstraction Level** | Community graph structure (flat → VMs; dense mesh → containers; tree/star → serverless) |
| **Resource Allocation** | IP/MAC/port churn, session duration from node roles |
| **Startup Time** | Latency spikes in bursty communities; role = “cold starter” |
| **Performance** | RTT, jitter, retransmissions — aggregate by community |
| **Isolation** | Role diversity in same subnet; reuse of IPs/MACs across communities |
| **State Management** | Roles showing persistence (e.g., DB servers) vs. stateless clients |
| **Scaling** | Flow bursts, dynamic community size, rapid IP churn |
| **Management Overhead** | Presence of mgmt traffic (e.g., kube-api, SSH); control-plane roles detected |
| **Cost Model** | Flow lifetimes, idle/active ratio, usage frequency by role and community |

### **Bare Metal Servers** provide direct access to physical hardware and full control over server infrastructure. From an application perspective bare metal servers offer the highest performance, the lowest overhead and the highest level of security isolation, but require the most manual management**.** Bare metal servers are used for high-performance computing, applications with specific hardware requirements and workloads needing consistent, predictable performance.

### **Virtual Machines (VMs)** provide a complete virtualized operating system using a hypervisor for isolated environments with dedicated resources. Operators can run multiple different OS types on the same physical hardware with full OS-level control, moderate overhead and resource consumption. Virtual server are used to run multiple different application environments, for legacy application hosting and scenarios that require full OS customization

### **Mini-VMs (Lightweight VMs) have a** smaller footprint compared to traditional VMs and faster startup times with less resource overhead. Employing hardware (chip-based) virtualization, they only require a hypervisor for partial OS virtualization and are more efficient than full VMs, but less efficient than containers. Lightweight VMs are used for microservices with moderate isolation requirements, for development and testing environments and applications needing more isolation than containers

### **Containers are** lightweight user space representations that share a single host OS kernel and enable rapid deployment and scaling with a consistent environment across development and production. Containers require minimal resource overhead and can rely on standardized packaging of applications. Containers are used in distributed or ‘cloud-native’ environments and allow for **c**ontinuous integration/continuous deployment (CI/CD).

### **Orchestrated Containers** rely on a central intelligence in a multi-server cluster environment for automated container management, dynamic scaling and load balancing and self-healing capabilities. The orchestrator abstracts complex networking and and provides service discovery capabilities to enable centralized container deployment and management. Orchestrated containers are ised for **l**arge, distributed microservices architectures that empower highly scalable applications in complex, dynamic computing environments.

### **Serverless Functions** enableephemeral, event-driven execution of containers with limited execution time and resources. Functions are extremely lightweight and fast and run with zero server management, scale automatically and are usually offered with a pay-per-execution pricing model. Functions are used for event-driven processing, intermittent or unpredictable workloads and microservices with specific, short-running tasks

From a provider perspective bare metal servers require physical servers dedicated to a single tenant. Virtual machines (VMs) a delivery environment for software-based emulation of physical hardware. Containers lightweight, portable execution environments that share the host OS kernel, orchestrated containers and orchestration platforms like Kubernetes and serverless functions and event-framework that enables stateless execution. The choice between these technologies depends on factors such as communication patterns, performance needs, and management overhead.

## 

## **Workload Optimization**

We look at the different artifact types to distinguish workloads that we run on dedicated, private infrastructure versus distributed, shared (cloud) environments. To determine the right target we classify artifacts that carry workloads in two dimensions, the consumption profile and the need for data sovereignty.

### Consumption Profile

**Variable Workloads -** Distributed systems, hence public clouds excel at handling fluctuating workloads due to their scalability. Organizations can use public cloud resources for peak demands and scale down during lulls, avoiding the cost of maintaining excess on-premises capacity. This "pay-as-you-go" model is ideal for applications with unpredictable traffic patterns.

**Stable Workloads -** Dedicated hardware runs predictable and consistent workloads more cost-effective in a private cloud or on-premises environment, because it requires less operational overhead. By keeping these workloads in-house, organizations avoid the ongoing costs of public cloud usage.

| **Trait** | **What to Look For in Data** | **Heuristic Have** |
| --- | --- | --- |
| **Variable Workloads**  (Cloud-friendly, bursty usage) | 🔸 High session volatility  🔸 Short lifespan (frame\_time\_epoch)  🔸 High TTL variability  🔸 Many external flows (flow\_relation)  🔸 Frequent port variation (many src\_port/dst\_port combinations)  🔸 MAC/IP reuse (many MACs per IP or vice versa) | ✔ session\_volatility  ✔ ttl\_variability  ✔ flow\_relation  ✔ IP-to-MAC mapping (ip\_reuse) |
| **Stable Workloads**  (On-prem/dedicated-friendly) | 🔸 Low session volatility  🔸 Consistent TTL  🔸 Mostly internal\_only flows  🔸 Long active duration per MAC+IP  🔸 Low MAC/IP reuse | ✔ is\_stable\_workload (you can derive)  ✔ flow\_relation analysis |

### Data Sovereignty

**Data-Intensive Workloads -** Storing and processing large volumes of data in the public cloud can be expensive. A hybrid approach allows organizations to keep data-intensive workloads in a private cloud, potentially reducing data transfer and storage costs.

**Compliance-Sensitive Workloads -** Maintaining sensitive data within a private cloud or on-premises environment can minimize compliance risks and associated costs. This approach allows organizations to leverage the public cloud for less sensitive data and applications.

| **Trait** | **What to Look For** | **Heuristic Add** |
| --- | --- | --- |
| **Data-Intensive Workloads** | 🔸 High volume per MAC+IP  🔸 High byte count (requires frame\_len aggregation)  🔸 Repeated large flows between internal nodes | Add: data\_volume = df.groupby('mac\_ip\_combo')['frame\_len'].sum() |
| **Compliance-Sensitive Workloads** | 🔸 Internal flows only  🔸 MACs/IPs that never communicate with external endpoints  🔸 Long-lived services (low churn) | ✔ flow\_relation = 'internal\_only'  ✔ Long active durations |

### 