Week 12: Hybrid Cloud Architecture

NT524 — Cloud Architecture and Security

PhD. Nguyen Ngoc Tu

December 3, 2025

Outline

1. Hybrid Cloud: Multi-Control-Plane Architecture
2. Artifacts & Workload Portability & Autonomous Scaling
3. Data Plane & Control Plane in Multi-Cloud
4. Deploy & Management Frameworks in Multi-Cloud Architecture
5. Security & Industrial Perspective
6. Case Studies: Hybrid Cloud Failures
7. Practice Overview

Learning Objectives

By the end of this week, you should be able to:

- **Motivate** when and why to use hybrid cloud in terms of **flexibility**, **incremental migration**, **compliance**, and **cost / risk management**.
- **Explain and sketch** hybrid architectures: key components, connectivity models (VPN, Direct Connect, etc.), and integration patterns across multiple control planes.
- **Design** workloads for portability and basic DR, including RPO/RTO targets and DNS / routing–based failover between on-prem and public cloud.
- **Implement and debug** a small hybrid lab by:
  - building OpenStack–public cloud connectivity, and
  - testing failover using Ansible playbooks and health-checked DNS.

---

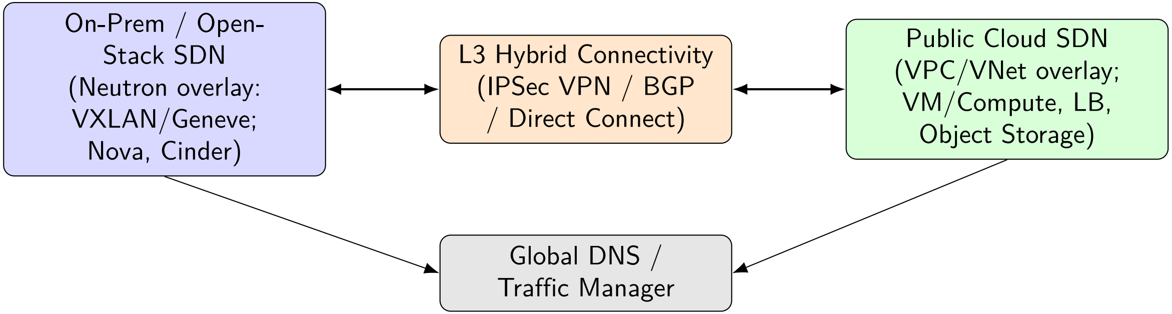
flexibility – linh hoạt; incremental migration – di trú từng bước; compliance – tuân thủ quy định; DR (Disaster Recovery) – khôi phục sau thảm hoạ

Motivation

- **Flexibility:** Run each workload in the *right place* (on-prem vs public cloud) based on performance, data locality, privacy and cost.
- **Incremental migration:** Use a *strangler* pattern to move services to the cloud gradually with lower risk.
- **Regulatory & risk:** Keep sensitive or regulated data on-prem or in specific regions, while bursting stateless workloads to public cloud.
- **Business continuity:** Design for DR and multi-site deployments to avoid a single-cloud *single point of failure*.

flexibility – linh hoạt; incremental migration – di trú từng bước; strangler pattern – mẫu strangler (thay thế dần); regulatory compliance – tuân thủ quy định; risk management – quản lý rủi ro; business continuity – duy trì liên tục kinh doanh; DR (Disaster Recovery) – khôi phục sau thảm hoạ; single point of failure – điểm lỗi đơn

Diagram: OpenStack SDN ↔ Public Cloud SDN (L3 Hybrid Connectivity)


On-Prem / Open-Stack SDN (Neutron overlay: VXLAN/Geneve; Nova, Cinder)
L3 Hybrid Connectivity (IPSec VPN / BGP / Direct Connect)
Public Cloud SDN (VPC/VNet overlay; VM/Compute, LB, Object Storage)
Global DNS / Traffic Manager

Note: VXLAN/Geneve overlays run *inside* SDN fabrics. Inter-cloud connectivity is always L3 (IPSec/BGP), not L2 bridging.

SDN (Software-Defined Networking) – mạng điều khiển bằng phần mềm; overlay network – mạng lớp phủ; VXLAN/Geneve – giao thức đường hầm VXLAN/Geneve; L3 connectivity – kết nối lớp 3 (IP); IPSec VPN – VPN IPSec; BGP (Border Gateway Protocol) – giao thức định tuyến biên

Why Multi-Control-Plane?

- Hybrid cloud is *not* just "connecting networks" with VPN/Direct Connect.
- Each environment (on-prem OpenStack, AWS/Azure/GCP, edge/Kubernetes) runs its own
  - **API surface** (OpenStack APIs, AWS/GCP/Azure, K8s),
  - **identity system** (Keystone, IAM, Azure AD, OIDC),
  - **policy engine** (RBAC, security groups, SCPs, Network Policies),
  - **resource lifecycle** (images, flavors/instance types, autoscaling, backup/DR rules).
- No single "global control plane" exists: attempts to *merge* them lead to fragile, tightly-coupled systems.
- Hybrid architecture must therefore:
  - **federate** identity and policy across planes (unify multiple independent systems),
  - use **shared artifacts** and GitOps/IaC as the common contract,
  - design clear **integration points** (DNS, VPN, APIs) between control planes.

API surface – giao diện lập trình; identity system – hệ thống định danh; policy engine – bộ máy thực thi chính sách; resource lifecycle – vòng đời tài nguyên; federation – liên kết liên miền; integration point – điểm tích hợp

What is a (Multi) Control Plane?

- A **control plane** is the layer that turns *intent* into real infrastructure and running workloads:
  - **provisioning** compute, network, storage,
  - **scheduling** workloads onto available capacity,
  - **policy enforcement** (security, quotas, placement),
  - managing **identity & access**,
  - collecting **telemetry** for autoscaling and health decisions.
- A **multi-control-plane** architecture means several independent control planes operate simultaneously, each with its own API, IAM model, policy framework and resource lifecycle.
- Examples across layers:
  - **On-prem**: OpenStack (Keystone, Nova, Neutron), VMware vCenter.
  - **Public cloud**: AWS (IAM + Organizations/Control Tower), Azure (ARM + Azure Policy), GCP (Resource Manager + Org Policy).
  - **OCI-based application orchestration**: container orchestrators (K8s, Nomad, ECS), artifact-driven deployment controllers (GitOps/ArgoCD/Flux), serverless runtimes (Lambda/Cloud Functions).
- Hybrid design focuses on making these planes *coordinate* via shared artifacts, identity federation, network connectivity, and DNS — do not merging them into a single system.

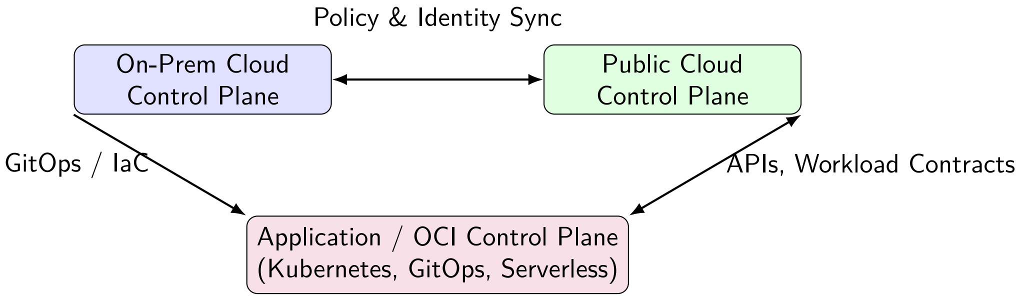
The Hybrid Problem

- **Control planes cannot be merged**: each platform (on-prem, cloud, OCI orchestrators) keeps its own release cycle, APIs and operational model. In practice we can only **federate** them.
- **Independent evolution**: new features, deprecations and limits appear at different times in each environment, so hybrid designs must tolerate version skew and partial support.
- **Cross-boundary alignment**: identities, roles, policies and governance rules must be mapped and kept consistent across:
  - IAM systems (Keystone, IAM, Azure AD, etc.),
  - policy engines (RBAC, SCP, Org Policy, NetworkPolicy),
  - compliance and audit requirements.
- **Network is necessary but not sufficient**: a working VPN/Direct Connect only solves reachability; you still need:
  - coherent naming and DNS,
  - shared artifacts and deployment pipelines,
  - aligned security controls and observability across planes.

---

federated – liên kết; version skew – lệch phiên bản; governance – quản trị; IAM (Identity and Access Management) – quản lý định danh và truy cập; compliance – tuân thủ quy định

Multi-Control-Plane Model


Policy & Identity Sync
On-Prem Cloud Control Plane
Public Cloud Control Plane
GitOps / IaC
APIs, Workload Contracts
Application / OCI Control Plane (Kubernetes, GitOps, Serverless)

control plane – mặt phẳng điều khiển; sync – đồng bộ; GitOps – triển khai điều khiển qua Git; IaC (Infrastructure as Code) – hạ tầng dưới dạng mã; workload contract – hợp đồng vận hành ứng dụng; OCI – chuẩn container mở (Open Container Initiative)

Principles of Multi-Control-Plane Architecture

1. **Federate, don't merge, control planes:** let on-prem, public cloud and application control planes evolve independently, and connect them only via well-defined interfaces (identity, network, DNS, artifacts).
2. **Treat identity as the core integration surface:** use SSO / OIDC / SAML, group and role mapping, and workload identities so that access decisions are consistent across planes.
3. **Use GitOps + IaC as the source of truth:** all environments pull desired state from version-controlled configuration, avoiding "click-ops" drift between control planes.
4. **Define runtime contracts for workloads:** standardise how apps consume config/secrets, expose health checks and metrics, and express scaling/placement needs across platforms.
5. **Enforce policy-as-code globally:** encode security, compliance and guardrails in machine-checkable policies (RBAC, SCP, Org Policy, NetworkPolicy) that are tested, reviewed and versioned like code.

federate – liên kết; integration surface – bề mặt tích hợp; source of truth – nguồn chân lý; drift – lệch cấu hình; runtime contract – hợp đồng vận hành ứng dụng; policy-as-code – chính sách dưới dạng mã

Policy Federation

- **Goal:** users and workloads should see *equivalent rules* regardless of which control plane they touch (on-prem, cloud, app).
- Policies must be aligned across:
  - **Resource access:** who can create/modify/delete which projects, VMs, clusters, buckets.
  - **Security & compliance:** encryption requirements, key usage, data residency, logging, retention.
  - **Naming, tagging, cost visibility:** consistent labels for environment (dev/test/prod), owner, cost-centre across all clouds.
  - **Network segmentation:** which tiers may talk to each other, and under what conditions (mTLS, firewall, SG/NSG rules).
- **Policy-as-code toolchain examples:**
  - **OPA / Gatekeeper** for Kubernetes and API admission controls.
  - **HashiCorp Sentinel** for Terraform and Vault policies.
  - **Cloud-native guardrails:** AWS SCP and Config Rules, Azure Policy / Blueprints, GCP Org Policy / Config Validator.
- **Federation pattern:** express global intent once (policy repo), then compile/translate into service-specific rules per control plane.

policy federation – liên kết chính sách; segmentation – phân đoạn mạng; guardrails – rào chắn chính sách; policy-as-code – chính sách dưới dạng mã; tagging – gán nhãn tài nguyên

Identity Federation

- **Identity is the unifying layer** across control planes: it decides who (user, service, workload) can do what and where.
- **Human identity federation:**
  - Use **OIDC/SAML federation** from a central IdP (Azure AD, Keycloak, Okta, etc.) into cloud IAM and on-prem platforms (OpenStack Keystone, GitLab, Kubernetes API).
  - Map groups/claims to roles and projects so the same user sees consistent permissions across environments.
- **Workload / service identity:**
  - Cloud-native: **IRSA** (AWS IAM Roles for Service Accounts), **GCP Workload Identity**, Azure Managed Identities.
  - Cross-environment: **SPIFFE/SPIRE** to issue short-lived identities (SVIDs) to services for mTLS and authorization.
- **Security principle:** avoid static credentials and long-lived access keys; prefer short-lived tokens, automatic rotation and identity-based access (no embedded secrets in images or code).

identity federation – liên kết danh tính; IdP (Identity Provider) – nhà cung cấp danh tính; workload identity – danh tính của ứng dụng/dịch vụ; static credential – khoá/tài khoản tĩnh; short-lived token – mã truy cập sống ngắn

Coordinated Automation

- **Goal:** all control planes (on-prem, cloud, application) are updated through the *same* automation story, not ad-hoc scripts.
- **GitOps as the backbone:** desired state for infra and workloads lives in Git; changes are reviewed, approved and rolled out via pipelines, not click-ops.
- **Tooling roles:**
  - **Terraform / Crossplane** for infrastructure (networks, VPCs, clusters, gateways, storage).
  - **ArgoCD / Flux** for workload deployment and promotion across clusters/environments using OCI images and manifests.
  - **Ansible** for VM- / OS-level configuration (e.g., strongSwan IPSec, system hardening, legacy services).
- **Outcome:**
  - declarative, auditable changes across all planes,
  - automatic reconciliation to the Git-defined state,
  - minimal configuration *drift* between environments (and a clear place to see and fix it when it happens).

---

declarative – kiểu khai báo; automation – tự động hoá; configuration drift – lệch cấu hình; click-ops – thao tác cấu hình bằng chuột (UI); reconciliation – đồng bộ trạng thái mong muốn

Summary: Multi-Control-Plane Mindset

- Hybrid cloud is about coordinating **control planes**, not merging them.
- Identity, policy, and automation are the glue.
- Workloads must rely on portable runtime contracts.
- GitOps orchestrates updates across environments.
- Connectivity supports the system—but governance shapes it.

Governance = Quản trị; Orchestrate = Điều phối

Artifacts & Workload Portability & Autonomous Scaling

Portable Workloads and Self-Adapting Cloud Applications

Artifact = Tạo phẩm phần mềm; Portability = Khả năng di động

What is an Artifact?

- An **artifact** is an *immutable, versioned output* of a build or authoring process that can be promoted across environments (on-prem, public cloud, edge) without modification.
- **Infrastructure & platform artifacts:**
  - **VM / image artifacts:** OpenStack Glance golden images, AMIs, cloud-init bundles.
  - **Network & infra templates:** Terraform modules, Heat/ARM/Bicep templates, Helm charts for infra components.
  - **Config & hardening baselines:** Ansible roles/playbooks, CIS-hardened OS profiles, strongSwan profiles.
- **Application-level artifacts:**
  - **OCI container images** and multi-arch image indexes.
  - **Deployment descriptors:** Helm charts, Kustomize overlays, Kubernetes manifests, serverless templates.
  - **SBOMs & policy bundles:** software bill of materials, OPA/Kyverno/Sentinel rule packs attached to the app.
- **Why artifacts matter in hybrid cloud:**
  - the same artifacts (images, network templates, manifests) can be applied by different control planes, ensuring *predictable behavior* across sites;
  - autoscaling, DR and failover reuse *approved* artifacts, reducing configuration drift and surprise differences between on-prem and public cloud.

Why Workload Portability?

- **Reduce vendor lock-in:** design workloads so they can move between on-prem and different public clouds without a full rewrite.
- **Placement flexibility:**
  - choose where to run (on-prem vs cloud, region A vs region B) based on cost, latency, data residency and capacity;
  - respond to price changes, new instance types or license models without being trapped on one platform.
- **Multiple runtime targets:**
  - **VM-based:** OpenStack, EC2, Azure VM, GCE.
  - **Kubernetes clusters:** on-prem, managed (EKS/AKS/GKE), edge clusters.
  - **Serverless platforms:** Lambda, Cloud Functions, Azure Functions, Knative.
- **Resilience & DR:**
  - the same workload can fail over to another region or cloud if a provider, region or data centre has an outage;
  - DR tests become "re-deploy artifact elsewhere" instead of "rebuild everything from scratch".
- **Hybrid/multi-cloud as the default:** portability is not optional if you want to use on-prem + cloud together, or to mix several clouds safely over time.

workload portability = khả năng di chuyển khối lượng công việc; vendor lock-in = khoá phụ thuộc nhà cung cấp;

The Runtime Contract

- A portable workload must obey a clear **runtime contract** so different platforms can run it without custom glue for each one:
  - **Packaging format: OCI image** (or VM image) as the standard unit of deployment, tagged by version/commit.
  - **Config & secrets interface:** configuration via environment variables and/or well-defined secret stores (Vault, cloud secret managers, K8s Secrets), not hard-coded in images.
  - **Observability contract:** consistent *metrics, logs, traces* exposed via OpenTelemetry or similar, so any platform can plug into its own monitoring/alerting stack.
  - **Network & service interface:** inbound traffic through stable API endpoints (DNS names, service names, virtual services) and—where used— service mesh / sidecar patterns for mTLS, retries and policies.
  - **Scaling signals:** expose health checks and load indicators (RPS, queue depth, latency) that autoscalers in any cloud can consume.
- **Once built, run anywhere:** CI produces signed, immutable artifacts; CD/GitOps reuses them across on-prem, multiple clouds and regions without rebuilding.

---

runtime contract – hợp đồng vận hành ứng dụng; packaging format – định dạng gói triển khai; config/secret interface – giao diện cấu hình và bí mật; instrumentation – gắn đo (thu thập số liệu); sidecar – mô-đun/bên phụ chạy kèm ứng dụng; autoscaler – bộ tự co giãn

Multi-Target Deployment (GitOps and Beyond)

- **Source of truth:** each environment should take desired state from a *single, versioned* source (often Git, but also config repositories, service catalogs, or artifact registries).
- **One artifact, many targets:**
  - build VM images / OCI images / manifests once,
  - deploy the *same* artifact to:
    - on-prem clusters or VM groups,
    - managed K8s (EKS/AKS/GKE),
    - edge or regional clusters / sites.
- **Environment-specific variants:**
  - base definitions capture common behavior,
  - overlays or parameters adjust per environment (URLs, resource limits, scale, credentials binding).
- **Implementation patterns:**
  - **GitOps controllers** (ArgoCD / Flux) that pull state from Git and reconcile clusters.
  - **CI/CD pipelines** (GitLab CI, GitHub Actions, Azure DevOps) that push releases to multiple targets.
  - **Fleet/orchestration tools** (Crossplane, Rancher Fleet, Spinnaker, config management DBs) coordinating many clusters/sites.
- **Hybrid benefit:** multi-target deployment ensures consistent rollout and rollback across on-prem and multi-cloud, regardless of which specific tool is used as the "control lever".

Workload Identity for Portability

- **Idea:** the *workload* (pod, VM, function) carries its own identity, instead of shipping API keys inside images or config files.
- **Cloud-native patterns:**
  - **OIDC-based identities** for services calling external APIs.
  - **IRSA** (IAM Roles for Service Accounts) for AWS EKS: pods receive short-lived credentials via service accounts.
  - **GCP Workload Identity**: bind K8s service accounts to GCP service accounts without long-lived keys.
  - **Azure Managed Identity**: workloads authenticate to Azure services without embedded secrets.
- **Cross-cloud / hybrid patterns:**
  - **SPIFFE/SPIRE** to issue uniform, short-lived identities (SVIDs) for services across clusters, clouds and on-prem.
  - mTLS between services based on these identities, not IPs.
- **Portability & security benefits:**
  - no hardcoded credentials in images, code or IaC,
  - easy key rotation and revocation,
  - the same workload spec can run on-prem or in different clouds by re-binding its identity, not changing its code.

Autonomous Scaling: Key Concepts

- **Autonomous scaling** = the application + platform *continuously adjust capacity* without human intervention based on live signals:
  - request load (RPS, queue depth),
  - resource usage (CPU, memory, I/O, connections),
  - latency and SLO/error-budget consumption.
- **Scaling patterns:**
  - **Horizontal scaling:** add/remove replicas, instances, or pods to change total capacity.
  - **Vertical scaling:** resize instances/pods with more CPU/RAM or faster storage; limited by single-node ceilings.
  - **Predictive scaling:** forecast future load from historical time-series and scale *before* spikes arrive.
- Good designs combine these patterns with cooldowns, hysteresis and headroom to avoid oscillation and overreaction.

autonomous scaling – tự co giãn tự động; SLO (Service Level Objective) – mục tiêu mức dịch vụ; error budget – ngân sách sai lỗi; horizontal scaling – mở rộng ngang; vertical scaling – mở rộng dọc; predictive – dự đoán trước

Kubernetes Autoscaling Stack

- **HPA (Horizontal Pod Autoscaler):**
  - scales the *number of pods* in a Deployment/ReplicaSet,
  - based on CPU/memory or custom metrics (via metrics-server/Prometheus).
- **VPA (Vertical Pod Autoscaler):**
  - recommends or applies new CPU/RAM requests/limits per pod,
  - useful for right-sizing long-running services.
- **Cluster Autoscaler:**
  - adds/removes worker nodes in the cluster,
  - integrates with cloud APIs (EC2/VMSS/MIG) or on-prem node pools.
- **KEDA (Kubernetes Event-Driven Autoscaling):**
  - scales workloads based on *external events* and queues: Kafka, RabbitMQ, SQS, Prometheus, HTTP rate, etc.
  - bridges classic HPA with rich, event-driven signals.

autoscaler – bộ tự co giãn; event-driven – dựa trên sự kiện; node pool – nhóm nút; right-sizing – định cỡ phù hợp

AI-Enhanced & Predictive Scaling

- **Predictive autoscaling** uses historical time-series (RPS, CPU, queue depth, time-of-day patterns) plus ML models to estimate future demand.
- **Cloud implementations:**
  - **AWS Predictive Scaling** for Auto Scaling Groups.
  - **Azure ML-based autoscale** integrated with VMSS and App Service.
  - **GCP** features in Autopilot / autoscaler that anticipate periodic or trend-driven load.
- **Benefits:**
  - reduces **cold starts** by scaling out ahead of traffic spikes,
  - lowers **overprovisioning** by shrinking capacity during predictable low periods,
  - smooths capacity adjustments, avoiding sharp oscillations.
- Works best when combined with SLO/error-budget policies and safeguards (max/min capacity, budget limits).

time-series – chuỗi thời gian; cold start – khởi động lạnh; overprovisioning – cấp dư tài nguyên; error budget – ngân sách sai lỗi

Portability for Hybrid DR

- **Goal:** the same workload can run on-prem or in cloud during failover, without code changes.
- **Single artifact set:**
  - OCI image + Helm chart / Kustomize base,
  - environment-specific overlays for on-prem vs cloud.
- **DR workflow:**
  - on-prem and cloud clusters kept in sync via GitOps or CI/CD,
  - a global load balancer or DNS (Route 53 / Traffic Manager / Cloud DNS) selects the active site based on health checks,
  - storage/state replicated by:
    - **block-level** replication (DRBD, storage arrays, volume replication),
    - **database-level** replication (CDC, logical replication, distributed SQL),
    - **object-storage** cross-region / cross-cloud mirroring.
- **Result:** fast failover, consistent workload identity and policy, and minimal manual intervention during incidents.

portability – khả năng di chuyển; DR (Disaster Recovery) – khôi phục sau thảm hoạ; failover – chuyển đổi dự phòng; replication – sao chép dữ liệu; CDC (Change Data Capture) – bắt thay đổi dữ liệu

Hybrid-Aware Predictive Scaling

- **Cloud-side capabilities:**
  - AWS Predictive Scaling for Auto Scaling Groups,
  - Azure ML-based autoscale policies,
  - GCP Autopilot / autoscaler predictive models.
- **Hybrid nuance:** treat on-prem as the *primary* site, cloud as *burst* capacity:
  - scale primarily on-prem until cluster or data-plane limits (CPU, memory, network, storage IOPS) are approached,
  - only trigger cloud scale-out when:
    - on-prem utilization stays above a threshold,
    - latency/SLO degradation is detected,
    - egress cost constraints are checked,
    - data-gravity / compliance rules allow placing data in cloud.
  - optionally pre-warm cloud capacity before predicted peaks.
- **Architecture pattern:** this implements a controlled **cloud burst** model — cloud is an elastic extension of on-prem, not the default execution environment.

data gravity – trọng lực dữ liệu; burst capacity – dung lượng bùng nổ tạm thời; pre-warm – khởi động sẵn; hybrid-aware – nhận thức ngữ cảnh lai (on-prem + cloud)

Summary: Portability & Autonomous Scaling

- Build once $\rightarrow$ deploy anywhere with OCI + GitOps.
- Runtime contracts guarantee portability.
- Workload identity eliminates environment-specific secrets.
- Autoscaling ensures efficiency, resilience, and cost savings.
- Predictive scaling further optimizes multi-cloud performance.

Resilience = Khả năng chống chịu; Efficiency = Hiệu quả

Data Plane vs. Control Plane

Defined Network Architecture in Hybrid Cloud

Data Plane = Mặt phẳng dữ liệu; Control Plane = Mặt phẳng điều khiển

What is the Control Plane?

- Manages the system:
  - API calls,
  - orchestration,
  - identity & access management,
  - policy enforcement,
  - monitoring & metrics.
- Low traffic volume, high criticality.

Orchestration = Điều phối; Enforcement = Thực thi

What is the Data Plane?

- Handles actual workload traffic:
  - service-to-service calls,
  - user requests,
  - database queries,
  - file transfer, streaming.
- High throughput, directly impacts performance.

Throughput = Lưu lượng xử lý

Why the Separation Matters?

- Control plane failure = **cluster is alive but unmanageable**.
- Data plane failure = **services unreachable or degraded**.
- Enables:
  - security zoning,
  - traffic isolation,
  - resilient network design,
  - scalable architecture.

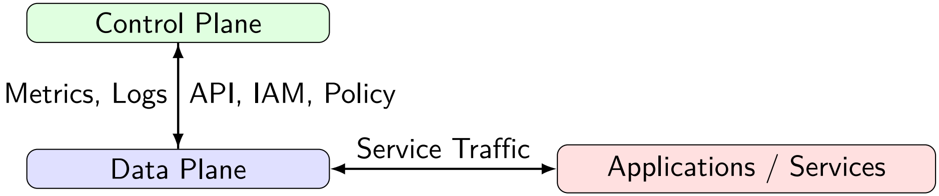
Zoning = Phân vùng; Resilient = Chịu lỗi tốt

Network Architecture Overview

- Hybrid cloud requires a **layered network architecture**:
  - Transport layer (connectivity),
  - Control plane API communication,
  - Data plane service traffic,
  - Policy + Identity alignment.
- Define boundaries explicitly.

Boundary = Biên; Layered = Phân lớp

Control vs. Data Plane Traffic



Metrics = Chỉ số; IAM = Quản lý truy cập định danh

Control Plane Connectivity Requirements

- Secure, low-latency links:
  - IPSec / WireGuard / TLS,
  - PrivateLink / VPC Endpoint,
  - Bastion-free API access via identity federation.
- Highly available DNS + service discovery.
- Strict access control + audit trail.

Bastion = Máy nhảy; Audit trail = Nhật ký kiểm toán

Data Plane Connectivity Requirements

- High-throughput, scalable paths:
  - Transit Gateway / Virtual WAN,
  - Direct Connect / ExpressRoute / Interconnect,
  - SD-WAN for branch or edge.
- Traffic engineering:
  - load-balancing,
  - mTLS service mesh,
  - latency-aware routing.

mTLS = Mã hoá TLS 2 chiều; Routing = Định tuyến

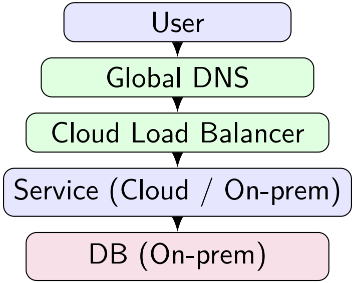
Example: Kubernetes Architecture

- Control Plane:
  - API Server,
  - Controller Manager,
  - Scheduler,
  - etcd datastore.
- Data Plane:
  - kubelet (partial),
  - container runtime,
  - service traffic between pods,
  - CNI-driven overlay networks.

CNI = Giao diện mạng vùng chứa

Hybrid Flow Example: Control vs. Data Plane

- Typical hybrid request path:



- **Control plane traffic:**
  - DNS resolution,
  - LB configuration APIs,
  - service discovery.
- **Data plane traffic:**
  - user request → service,
  - service → DB query,
  - largest source of latency & egress cost.

Cross-Cloud Service Mesh in Hybrid Data Plane

- Service mesh extends security & routing to multi-cloud:
  - **Istio multi-cluster**,
  - **Linkerd multi-cluster**,
  - **Consul mesh**.
- Capabilities:
  - mTLS across clouds,
  - traffic shaping (canary, failover),
  - unified observability,
  - identity-based routing (SPIFFE).
- Benefit: a consistent **data plane fabric** across on-prem + cloud.

Service Mesh = Lưới dịch vụ; Fabric = Kết cấu đồng nhất

Summary: Defined Network Architecture

- Control Plane = orchestration, policy, identity.
- Data Plane = workload traffic, user requests, DB access.
- Separation enables scalable, secure hybrid design.
- Hybrid architecture must define:
  - connectivity layers,
  - identity & policy flow,
  - traffic routing,
  - monitoring and governance.

Governance = Quản trị; Routing = Định tuyến

Deploy & Management Frameworks

Infrastructure Deployment and Cloud Management

Deploy = Triển khai; Framework = Khung làm việc

Why Deployment Frameworks Matter

- Hybrid cloud requires consistency across environments.
- Manual provisioning $\rightarrow$ drift, errors, security gaps.
- Frameworks automate:
  - infrastructure,
  - configuration,
  - compliance,
  - operational governance.
- Goal: reproducible, predictable, auditable environments.

Drift = Lệch cấu hình; Reproducible = Tái tạo được

Terraform: Infrastructure as Code

- HashiCorp Terraform = declarative IaC for hybrid/multi-cloud.
- Strengths:
  - provider ecosystem (AWS, Azure, GCP, OpenStack),
  - stateful planning,
  - modular architecture,
  - environment reproducibility.
- Used for: VPCs, VNets, VM pools, K8s nodes, gateways.

Declarative = Khai báo; Provider = Nhà cung cấp tích hợp

Terraform: Example Snippet

```
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "public" {
  vpc_id = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"
}
```

CIDR = Dải mạng phân lớp; Subnet = Mạng con

Ansible: Configuration as Code

- Agentless automation tool using SSH/WinRM.
- Best for:
  - OS configuration,
  - application setup,
  - package installation,
  - VPN configuration (strongSwan, WireGuard),
  - VM bootstrapping.
- Complements Terraform.

Agentless = Không dùng tác nhân; Bootstrap = Khởi tạo ban đầu

Terraform vs. Ansible

Terraform	Provision infrastructure (networks, VMs, clusters)
Ansible	Configure systems after they exist (services, packages, OS tuning)
Together	Complete lifecycle: Provision → Configure → Deploy

Provision = Cấp tài nguyên; Configure = Cấu hình

Public Cloud Deployment Frameworks

- **AWS:**
  - CloudFormation (declarative IaC),
  - CDK (IaC with programming languages),
  - Systems Manager (run commands, patching).
- **Azure:**
  - ARM Templates / Bicep,
  - Azure Automation / DSC.
- **GCP:**
  - Deployment Manager,
  - Config Controller (Anthos + GitOps).

Template = Mẫu; Automation = Tự động hoá

Kubernetes as a Multi-Cloud Orchestrator

- Kubernetes provides a consistent API across environments.
- Tools:
  - **Crossplane**: provision cloud infra via CRDs,
  - **ArgoCD**: GitOps deployment,
  - **Flux**: continuous delivery & image automation.
- Pattern: Use K8s as the unified control surface.

CRD = Định nghĩa tài nguyên mở rộng; GitOps = Triển khai qua Git

GitOps Deployment Pipeline

- Repository = single source of truth.
- Pipeline stages:
  1. Developer commits → triggers CI.
  2. CI builds & pushes artifact.
  3. GitOps tool detects change.
  4. ArgoCD/Flux applies manifests to cluster.
  5. Drift detection ensures compliance.

Source of truth = Nguồn gốc chuẩn; Drift detection = Phát hiện lệch cấu hình

Summary: Deploy & Management Frameworks

- Terraform $\rightarrow$ Infrastructure provisioning across clouds.
- Ansible $\rightarrow$ System configuration and orchestration.
- Cloud-native frameworks $\rightarrow$ deeper integration with each provider.
- Kubernetes (Crossplane + ArgoCD) $\rightarrow$ unified control for hybrid/multi-cloud.
- Combined $\rightarrow$ Automated, compliant, resilient deployments.

Resilient = Chịu lỗi; Compliant = Tuân thủ

Security & Industrial Perspective

Security Architecture and Real-World Industrial Practice

Industrial Perspective = Góc nhìn công nghiệp

Why Security is Different in Hybrid Cloud

- Multiple trust domains (on-prem, cloud, edge).
- Heterogeneous identity systems.
- Varying network boundaries and policies.
- Increased attack surface due to interconnectivity.
- Need for shared responsibility clarity.

Attack surface = Bề mặt tấn công; Trust domain = Miền tin cậy

Zero-Trust as Industrial Standard

- Industry is shifting from perimeter defense → zero-trust.
- Core ideas:
  - verify every connection,
  - continuous authentication,
  - context-aware authorization,
  - microsegmentation in network & service mesh.
- Enforced by identity, not IP address.

Zero-Trust = Không tin cậy mặc định; Microsegmentation = Vi phân đoạn

Industrial Security Controls

- **Identity-first security:**
  - OIDC federation, SSO, MFA.
- **Data protection:**
  - encryption at rest & in transit,
  - HSM/KMS-backed key lifecycle.
- **Network security:**
  - microsegmentation,
  - service mesh mTLS,
  - cloud firewalling + eBPF filters.

HSM = Phần cứng mã hoá; mTLS = TLS hai chiều

Threat Modeling in Industry

- Standard frameworks:
  - STRIDE,
  - MITRE ATT&CK,
  - Kill-chain analysis.
- Hybrid cloud attack vectors:
  - misconfigured VPNs,
  - insecure API endpoints,
  - exposed service credentials,
  - lateral movement across environments.

Attack vector = Hướng tấn công; Lateral movement = Di chuyển ngang

Compliance in Real Deployments

- Industry must meet regulatory requirements:
  - ISO 27001, PCI-DSS, HIPAA,
  - GDPR / Data residency,
  - National cybersecurity laws.
- Hybrid cloud must enforce:
  - logging & auditing,
  - retention policies,
  - data sovereignty controls.

Residency = Lưu trú dữ liệu; Sovereignty = Chủ quyền dữ liệu

Security Automation in Industry

- CI/CD integrated with:
  - SAST (static code scans),
  - DAST (runtime scans),
  - IaC security scanning,
  - container vulnerability scanning.
- Runtime controls:
  - Falco / eBPF,
  - Cloud IDS/IPS,
  - WAF + API security gateways.

SAST/DAST = Quét bảo mật mã nguồn/chạy thử

Observability as a Security Tool

- Unified telemetry pipeline:
  - OpenTelemetry (traces/metrics/logs),
  - SIEM (Splunk, ELK, Sentinel),
  - CloudTrail, Audit Logs.
- Detection via:
  - anomaly detection,
  - behavior-based alerts,
  - workload identity violations.

SIEM = Hệ thống giám sát sự kiện bảo mật; Anomaly = Bất thường

Industrial Architecture Patterns

- **Pattern 1:** Cloud landing zones with identity federation.
- **Pattern 2:** Segmented network zones (DMZ, App, Data).
- **Pattern 3:** Multi-layered firewalls (network + service mesh).
- **Pattern 4:** GitOps + policy-as-code for governance.
- **Pattern 5:** DR tested via automated chaos drills.

Landing zone = Vùng triển khai chuẩn; Governance = Quản trị

Summary: Industrial Perspective

- Modern security = identity-first + zero-trust.
- Industry focuses on automation, observability, and policy guardrails.
- Hybrid cloud increases complexity → requires layered defense.
- Compliance and governance shape real deployments.
- Security must be continuous, measurable, and automated.

Guardrail = Rào chắn; Continuous = Liên tục

Case Studies: Hybrid Cloud Failures

Industrial Incidents and Lessons Learned

Case Study = Nghiên cứu tình huống; Incident = Sự cố

Case Study 1: Misconfigured VPN Tunnel

- Enterprise hybrid cloud using IPSec tunnels.
- Incorrect phase-2 proposal → tunnel flapped every few minutes.
- Result:
  - intermittent connectivity,
  - service timeouts,
  - cascading failures in microservices.
- Root cause: no automated config validation or monitoring.

Flapped = Chập chờn; Cascading failure = Lỗi dây chuyền

Case Study 2: Unexpected Egress Cost Spike

- Analytics platform split across on-prem + cloud.
- Large data pulls from cloud $\rightarrow$ on-prem DB every night.
- Monthly cost increased by 300%.
- Causes:
  - no egress monitoring,
  - poor compute-to-data locality,
  - misaligned cluster placement.
- Result: emergency budget freeze + architecture rewrite.

Egress = Lưu lượng ra; Locality = Tính cục bộ

Case Study 3: Identity Federation Failure

- Company used OIDC federation for hybrid access.
- IdP certificate expired $\rightarrow$ cloud APIs became inaccessible.
- Effects:
  - deployments failed,
  - autoscaling stalled,
  - emergency rollback took hours.
- Root cause: no certificate rotation automation.

Federation = Liên kết; Certificate rotation = Xoay vòng chứng thư

Case Study 4: DR Plan Failed During Real Outage

- DR designed for “pilot-light” cloud region.
- Failover runbook never tested end-to-end.
- During outage:
  - data replication lagged by 6 hours,
  - scripts broke due to outdated resource names,
  - load-balancer policies mismatched.
- Result: 12-hour downtime.

Pilot-light = Chế độ đèn mồi; Runbook = Quy trình ứng cứu

Lessons Learned from Industry

- Test hybrid infrastructure continuously:
  - automated DR drills,
  - config drift detection.
- Enforce identity/secret lifecycle automation.
- Monitor egress, latency, and cross-cloud dependencies.
- Adopt GitOps + policy-as-code for consistency.
- Design for failure: redundancy, region independence, chaos testing.

Drill = Diễn tập; Redundancy = Dư thừa

Management Challenges & Cost

- **Mạng:** MTU mismatch, route asymmetry, overlapping CIDR, ACL lệch.
- **Vận hành:** drift cấu hình giữa môi trường; thiếu chuẩn tag/CMDB.
- **Chi phí:** egress cross-cloud, nhân đôi dịch vụ (SIEM/backup), đường chuyên dụng.
- **Giảm thiểu:** chuẩn hoá IaC, central policy, FinOps (budget/alert/showback), kiến trúc data locality.

Practice Overview

- **Task 1:** Tạo mạng OpenStack "onprem" (10.20.0.0/16) và AWS VPC "cloud" (10.30.0.0/16).
- **Task 2:** Thiết lập site-to-site IPSec (strongSwan on-prem ↔ AWS VPN).
- **Task 3:** Triển khai app đơn giản ở cả hai nơi; cấu hình DNS failover (Route 53).
- **Task 4:** Chạy thử failover: dừng app on-prem, xác thực chuyển hướng sang cloud.

Ansible: OpenStack Network + VM (On-Prem)

```
- hosts: controller
  tasks:
    - name: Create on-prem network + subnet
      openstack.cloud.network: { state: present, name: onprem-net }
    - openstack.cloud.subnet:
        state: present
        network_name: onprem-net
        name: onprem-subnet
        cidr: 10.20.1.0/24
        enable_dhcp: yes
    - name: Launch strongSwan gateway VM
      openstack.cloud.server:
        name: ipsec-gw
        image: "ubuntu-22.04"
        flavor: "m1.small"
        networks: [{ network: onprem-net }]
        key_name: "wk10-key"
```

Ansible: Configure strongSwan (On-Prem IPSec)

```
- hosts: ipsec_gw
  become: true
  vars:
    peer_public_ip: "AWS_VPN_PUBLIC_IP"
    local_net: "10.20.0.0/16"
    remote_net: "10.30.0.0/16"
    psk: "{{ vault_psk }}" # store in Ansible Vault
  tasks:
    - name: Install strongSwan
      package: { name: strongswan, state: present }

    - name: ipsec.conf
      copy:
        dest: /etc/ipsec.conf
        content: |
          config setup
            charondebug="ike 1, knl 1, cfg 1"
          conn aws
            keyexchange=ikev2
            authby=psk
            left=%any
```

Ansible: AWS VPC + VPN

```
- hosts: localhost
  connection: local
  vars:
    region: ap-southeast-1
    onprem_ip: "ONPREM_PUBLIC_IP"
  tasks:
    - name: Create VPC
      amazon.aws.ec2_vpc_net:
        name: cloud-vpc
        cidr_block: 10.30.0.0/16
        region: "{{ region }}"
        dns_hostnames: yes
        state: present
      register: vpc

    - name: Create customer gateway (on-prem)
      amazon.aws.ec2_customer_gateway:
        bgp_asn: 65010
        ip_address: "{{ onprem_ip }}"
        type: ipsec.1
        state: present
```

Ansible: Route 53 Failover DNS

```
- hosts: localhost
  connection: local
  vars:
    zone: "example.edu"
    record: "hybrid.app.example.edu"
    primary_ip: "ONPREM_PUBLIC_APP_IP"
    secondary_ip: "CLOUD_PUBLIC_APP_IP"
  tasks:
    - name: Create health check for on-prem
      community.aws.route53_health_check:
        fqdn: "{{ record }}"
        port: 80
        type: HTTP
        resource_path: "/healthz"
      register: hc

    - name: Create failover A records
      community.aws.route53:
        zone: "{{ zone }}"
        record: "{{ record }}"
        type: A
```

Operational Pitfalls

- **CIDR chồng lấp (overlap):** cần kế hoạch địa chỉ từ đầu; dùng NAT khi buộc phải.
- **MTU/Fragmentation:** IPsec thêm overhead; chỉnh `mtu` hoặc `MSS clamping`.
- **Asymmetric routing:** kiểm tra route tables, policy-based routing.
- **DNS TTL:** TTL cao $\Rightarrow$ failover chậm; TTL quá thấp $\Rightarrow$ chi phí/độ ổn định.
- **Egress cost:** đồng bộ dữ liệu qua cloud tốn phí; cân nhắc caching/edge/partitioning.

Mini Lab

1. Tạo OpenStack on-prem net + VM gateway; tạo AWS VPC.
2. Cấu hình IPSec (strongSwan $\leftrightarrow$ AWS VPN); ping qua lại hai CIDR.
3. Deploy app "hello" ở cả hai bên; tạo DNS failover bản ghi A.
4. Tắt app on-prem; quan sát chuyển hướng sang cloud; chụp màn hình kết quả.

Assessment & Deliverables

- **Báo cáo (2–3 trang):** kiến trúc hybrid, sơ đồ kết nối, RPO/RTO, kết quả failover.
- **Minh chứng:** ảnh chụp: VPN up, routing OK, DNS record, health check, failover.
- **Tệp nộp:** playbooks (OpenStack/AWS), cấu hình strongSwan, lưu đồ DR.
- **Rubric (10 điểm):** kết nối đúng (3), failover hoạt động (3), bảo mật/IaC chuẩn (2), tài liệu (2).

Threat Model Lite (STRIDE) cho Hybrid

- **Spoofing:** giả mạo gateway/API ⇒ mTLS, IP allowlist, IAM federation chặt.
- **Tampering:** thay đổi route/SG trái phép ⇒ IaC, change control, drift detection.
- **Repudiation:** thiếu audit ⇒ central logs, CloudTrail/Keystone audit.
- **Information Disclosure:** tunnel lộ thông tin ⇒ IPSec mạnh, rotate PSK/cert.
- **DoS:** saturate VPN/Direct Connect ⇒ QoS, rate-limit, HA tunnels.
- **EoP:** trust boundary mờ ⇒ microsegmentation, least privilege cross-cloud.

Wrap-up

- Hybrid = linh hoạt + tuân thủ + giảm rủi ro, nhưng đòi hỏi **thiết kế mạng/định danh/kỷ luật IaC**.
- DR cơ bản: hiểu RPO/RTO; chọn pattern phù hợp (pilot light/warm standby/active-active).
- Lab: thiết lập VPN, triển khai app hai nơi, cấu hình DNS failover và thử nghiệm.
- Tuần 11: **Cloud Observability & Incident Response**.