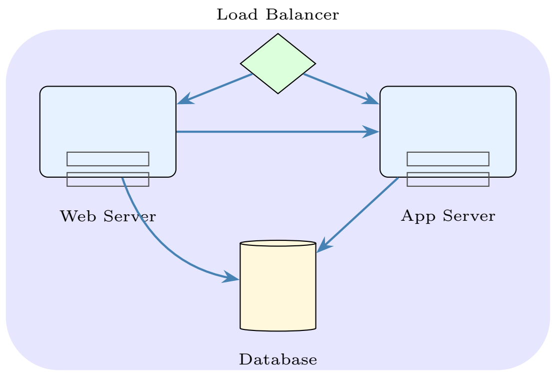
Deployment Viewpoint

Architecture Viewpoint Specification



Version:	2.0
Status:	Release
Classification:	ISO/IEC/IEEE 42010 Compliant
Last Updated:	December 12, 2025

Based on the Views and Beyond approach to software architecture documentation

Contents

1 Viewpoint Name

Viewpoint Identification

Name:	Deployment Viewpoint
Synonyms:	Installation Viewpoint, Physical Viewpoint, Infrastructure Viewpoint, Allocation Viewpoint, Deployment View, Physical Architecture View, Infrastructure View, Runtime Environment View
Identifier:	VP-DEPLOY-001
Version:	2.0

1.1 Viewpoint Classification

The Deployment Viewpoint belongs to the **Allocation Style** family within the Views and Beyond approach. It specifically addresses the mapping between software elements and the environmental structures (hardware, networks, file systems, and runtime platforms) in which the software executes. This viewpoint bridges the gap between logical software architecture and physical infrastructure architecture.

Table 1: Viewpoint Classification Taxonomy

Attribute	**Value**
Style Family	Allocation Styles
Primary Focus	Runtime Infrastructure Mapping
Abstraction Level	Physical/Environmental
Temporal Perspective	Execution Time
Related Styles	Work Assignment, Implementation, Install
IEEE 42010 Category	Environmental Viewpoint

2 Overview

The Deployment Viewpoint provides a comprehensive framework for documenting how software elements are allocated to hardware resources and execution environments. This viewpoint is essential for understanding, analyzing, and communicating the physical manifestation of a software system within its operational context.

2.1 Purpose and Scope

The primary purpose of this viewpoint is to establish a clear mapping between software artifacts (components, modules, processes, threads) and the hardware/infrastructure elements (servers, containers, virtual machines, network devices) on which they execute. This mapping is fundamental to understanding system behavior, performance characteristics, security boundaries, and operational concerns.

Viewpoint Definition

The Deployment Viewpoint defines the correspondence between software elements of the system architecture and elements of a platform or environment model in which the software executes. It shows how software is distributed across computing nodes and how these nodes are interconnected through network infrastructure.

2.2 Key Characteristics

The Deployment Viewpoint exhibits several distinctive characteristics that differentiate it from other architectural viewpoints:

Multi-dimensional Mapping: Unlike simpler viewpoints that focus on a single type of relationship, the Deployment Viewpoint captures multiple simultaneous mappings—software to hardware, processes to runtime environments, data to storage systems, and communication to network infrastructure.

Runtime Focus: This viewpoint is concerned exclusively with the system as it exists during execution, rather than during development or build time. It captures the dynamic allocation of software to resources and the relationships that exist when the system is operational.

Environmental Integration: The viewpoint explicitly incorporates environmental elements that are typically external to the software architecture itself, including operating systems, middleware platforms, container orchestration systems, and cloud infrastructure services.

Quality Attribute Implications: Deployment decisions have profound implications for numerous quality attributes including performance, availability, security, scalability, and maintainability. This viewpoint makes these implications explicit and analyzable.

2.3 Relationship to Other Viewpoints

The Deployment Viewpoint does not exist in isolation but has important relationships with other architectural viewpoints:

Table 2: Relationships to Other Viewpoints

Viewpoint	Relationship
Component-and-Connector	Software components identified in C&C views are the elements being deployed. Runtime connections inform network topology requirements.
Module	Module structure influences packaging and deployment unit boundaries. Build outputs become deployment artifacts.
Concurrency	Process and thread structures are allocated to execution environments. Concurrency strategies affect node allocation.
Information/Data	Data stores are mapped to storage infrastructure. Data flow paths influence network design.
Security	Security domains map to network segments. Trust boundaries align with deployment boundaries.
Operational	Deployment topology enables or constrains operational procedures. Recovery strategies depend on deployment architecture.

3 Concerns

This section enumerates the architectural concerns that the Deployment Viewpoint is designed to address. These concerns represent the fundamental questions that stakeholders have about the system's physical architecture and infrastructure.

3.1 Primary Concerns

C1: Hardware Resource Allocation

- What hardware resources (servers, storage, network devices) does the system require?
- How are software components allocated to specific hardware resources?
- What are the resource requirements (CPU, memory, storage, bandwidth) for each allocation?
- How do hardware capabilities constrain software functionality?

C2: Network Topology and Communication

- How are hardware nodes interconnected?
- What network protocols are used for communication between nodes?
- What are the bandwidth and latency characteristics of network connections?
- How does network topology support or constrain software communication patterns?

C3: Performance and Scalability

- How does deployment topology affect system performance?
- What are the bottlenecks in the current deployment configuration?
- How can the system scale horizontally and vertically?
- What load balancing and traffic distribution strategies are employed?

C4: Availability and Reliability

- How does the deployment architecture support high availability?
- What redundancy mechanisms are in place?
- How are failover and recovery procedures implemented?
- What are the single points of failure in the deployment?

C5: Security and Isolation

- How are security zones and trust boundaries implemented?
- What network segmentation strategies are employed?
- How is access to infrastructure resources controlled?
- What encryption is applied to data in transit and at rest?

C6: Operational Management

- How is the system deployed and updated?
- What monitoring and observability infrastructure is in place?
- How are logs collected and aggregated?
- What configuration management approaches are used?

C7: Cost and Resource Efficiency

- What are the infrastructure costs associated with the deployment?
- How efficiently are resources utilized?
- What opportunities exist for cost optimization?
- How do different deployment options compare in terms of total cost of ownership?

C8: Compliance and Regulatory Requirements

- What geographic constraints apply to data and processing?
- How does deployment satisfy regulatory requirements?
- What audit trails and compliance controls are implemented?
- How are data sovereignty requirements addressed?

3.2 Concern-Quality Attribute Mapping

The following table maps deployment concerns to the quality attributes they most directly influence:

Table 3: Concern to Quality Attribute Mapping

Concern	**Performance**	**Availability**	**Security**	**Scalability**	**Maintainability**	**Cost**
Hardware Allocation	●	○	○	●	○	●
Network Topology	●	●	●	●	○	○
Scaling Strategy	●	○	○	●	○	●
Redundancy	○	●	○	○	○	●
Security Zones	○	○	●	○	○	○
Operations	○	●	○	○	●	○

● = Primary impact, ○ = Secondary impact

4 Anti-Concerns

Understanding what the Deployment Viewpoint is *not* appropriate for is equally important as understanding its intended use. The following anti-concerns help stakeholders avoid misapplying this viewpoint.

4.1 Out of Scope Topics

AC1: Detailed Software Design

- Internal component algorithms and data structures
- Detailed class hierarchies and object relationships
- Method signatures and API contracts
- Code organization within individual components

AC2: Development Process Concerns

- Source code repository structure
- Build system configuration details
- Continuous integration pipeline design
- Development environment setup

AC3: Business Logic Specification

- Business rules and workflow definitions
- Domain model semantics
- User interface design and user experience
- Feature specifications and requirements

AC4: Data Schema Design

- Database schema definitions
- Entity-relationship models
- Data validation rules
- Query optimization strategies

AC5: Procurement and Vendor Selection

- Hardware vendor comparisons
- Cloud provider selection criteria
- Contract negotiations
- Licensing decisions

Common Misapplications

Avoid using the Deployment Viewpoint for:

- Documenting software module dependencies (use Module Viewpoint)
- Specifying runtime behavior and interactions (use C&C Viewpoint)
- Defining development team organization (use Work Assignment Viewpoint)
- Describing installation procedures (use Implementation/Install Viewpoint)
- Modeling business processes (use Process Viewpoint or BPMN)

5 Typical Stakeholders

The Deployment Viewpoint serves multiple stakeholder communities, each with distinct information needs and concerns. Understanding these stakeholders helps architects tailor deployment documentation appropriately.

5.1 Primary Stakeholders

Table 4: Primary Stakeholder Analysis

Stakeholder	Role Description	Primary Interests
Infrastructure Architects	Design physical and virtual infrastructure	Hardware selection, network design, capacity planning, technology standards
System Administrators	Deploy, configure, and maintain systems	Installation procedures, configuration management, monitoring setup, backup strategies
Network Engineers	Design and manage network infrastructure	Network topology, security zones, bandwidth allocation, routing configuration
Security Architects	Ensure security of deployed systems	Security zones, trust boundaries, encryption, access control, compliance
DevOps Engineers	Automate deployment and operations	CI/CD pipelines, container orchestration, infrastructure as code, automation
Cloud Architects	Design cloud-native solutions	Cloud service selection, multi-cloud strategy, serverless architecture, cost optimization

5.2 Secondary Stakeholders

Table 5: Secondary Stakeholder Analysis

Stakeholder	Role Description	Primary Interests
Software Architects	Design software structure and behavior	Component allocation, integration points, quality attribute implications
Development Teams	Build software components	Runtime environment, dependencies, testing environments, deployment targets
Performance Engineers	Optimize system performance	Resource allocation, bottleneck identification, capacity modeling
Project Managers	Plan and track delivery	Resource requirements, timeline impacts, risk identification
Operations Managers	Oversee production systems	Operational procedures, SLA compliance, incident management
Finance/Procurement	Manage infrastructure costs	Cost estimates, licensing requirements, vendor management
Compliance Officers	Ensure regulatory compliance	Data residency, audit trails, compliance controls

5.3 Stakeholder Concern Matrix

Table 6: Stakeholder-Concern Responsibility Matrix

	Hardware	Network	Performance	Availability	Security	Operations	Cost	Compliance
Infra. Architect	R	R	A	A	C	I	C	I
System Admin	A	C	I	R	I	R	I	I
Network Engineer	C	R	C	C	A	I	I	I
Security Architect	I	A	I	I	R	C	I	A
DevOps Engineer	C	C	C	A	C	R	I	I
Cloud Architect	R	A	A	A	A	C	R	C

R = Responsible, A = Accountable, C = Consulted, I = Informed

6 Model Types

The Deployment Viewpoint employs several complementary model types to capture different aspects of the deployment architecture. Each model type serves specific documentation purposes and addresses particular stakeholder concerns.

6.1 Model Type Catalog

MT1: Deployment Diagram

- *Purpose:* Show allocation of software elements to hardware nodes
- *Primary Elements:* Nodes, artifacts, deployment specifications
- *Key Relationships:* Deploys, manifests, communicates-with
- *Typical Notation:* UML Deployment Diagram, custom box-and-line

MT2: Network Topology Diagram

- *Purpose:* Illustrate network infrastructure and connectivity
- *Primary Elements:* Network devices, subnets, zones, connections
- *Key Relationships:* Connects-to, routes-through, contained-in
- *Typical Notation:* Network diagrams, zone diagrams

MT3: Infrastructure Stack Diagram

- *Purpose:* Show layered infrastructure components
- *Primary Elements:* Hardware, OS, middleware, runtime, application
- *Key Relationships:* Runs-on, depends-on, provides
- *Typical Notation:* Layered stack diagrams

MT4: Environment Specification

- *Purpose:* Document environment configurations
- *Primary Elements:* Environments, configurations, variables
- *Key Relationships:* Promotes-to, differs-from, configures
- *Typical Notation:* Tables, configuration specifications

MT5: Resource Allocation Model

- *Purpose:* Specify resource requirements and allocations
- *Primary Elements:* Resources, capacities, allocations, limits
- *Key Relationships:* Requires, allocates, limits
- *Typical Notation:* Tables, capacity models

MT6: Security Zone Diagram

- *Purpose:* Illustrate security boundaries and trust levels
- *Primary Elements:* Zones, boundaries, gateways, policies
- *Key Relationships:* Contains, protects, permits, denies
- *Typical Notation:* Zone diagrams, network security diagrams

MT7: High Availability Model

- *Purpose:* Document redundancy and failover mechanisms
- *Primary Elements:* Clusters, replicas, load balancers, failover paths
- *Key Relationships:* Replicates, fails-over-to, load-balances
- *Typical Notation:* HA topology diagrams

MT8: Container Orchestration Model

- *Purpose:* Document containerized deployment topology
- *Primary Elements:* Clusters, nodes, pods, services, ingress
- *Key Relationships:* Schedules, exposes, routes, scales
- *Typical Notation:* Kubernetes architecture diagrams

6.2 Model Type Relationships

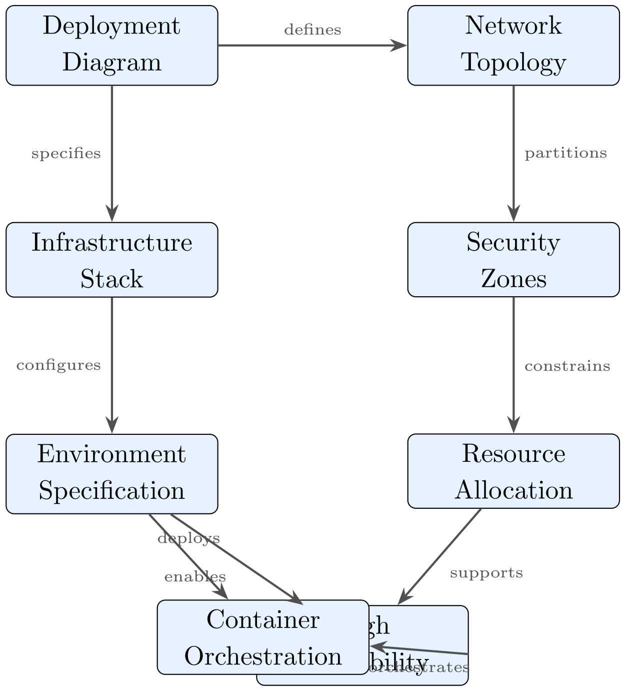


Figure 1: Model Type Dependency Relationships

7 Model Languages

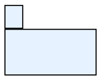
For each model type, specific languages, notations, and modeling techniques are prescribed. This section documents the vocabulary and grammar for constructing deployment views.

7.1 UML Deployment Diagram Notation

The Unified Modeling Language (UML) provides a standardized notation for deployment diagrams that is widely recognized and tool-supported.

7.1.1 Primary Notation Elements

Table 7: UML Deployment Diagram Notation Elements

Symbol	Element	Description
	Node	Physical or virtual computing resource capable of hosting software
artifact	Artifact	Deployable software element (executable, library, configuration)
———	Association	General relationship between elements
- - - -➤	Dependency	One element depends on another
▶———	Composition	Strong ownership relationship

7.1.2 Node Stereotypes

UML stereotypes extend the base notation to represent specific infrastructure types:

Table 8: Common Node Stereotypes

Stereotype	Description
`<<device>>`	Physical hardware device
`<<executionEnvironment>>`	Runtime environment (JVM, container, etc.)
`<<server>>`	Server machine (physical or virtual)
`<<database>>`	Database server or instance
`<<container>>`	Container runtime (Docker, etc.)
`<<pod>>`	Kubernetes pod
`<<cluster>>`	Cluster of nodes
`<<loadBalancer>>`	Load balancing device or service
`<<firewall>>`	Network firewall
`<<storage>>`	Storage system or volume
`<<cloud>>`	Cloud infrastructure service

7.2 Network Diagram Notation

Network topology diagrams employ industry-standard symbols and conventions for representing infrastructure.

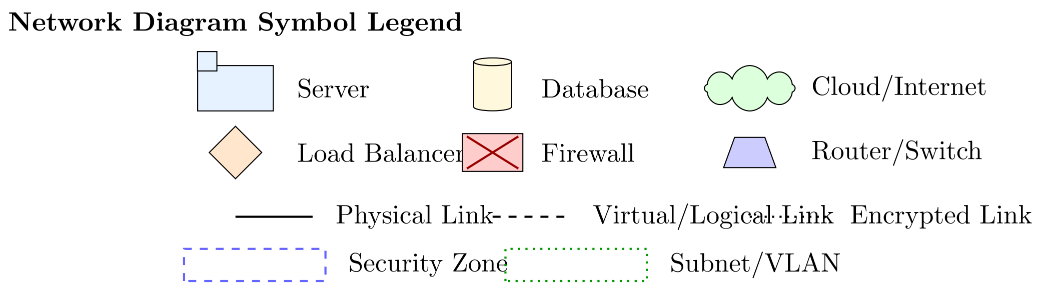


Figure 2: Network Diagram Symbol Legend

7.3 Infrastructure as Code Languages

Modern deployment documentation often includes or references Infrastructure as Code (IaC) specifications. The following languages are commonly used:

Table 9: Infrastructure as Code Languages

Language	**Platform**	**Use Case**
Terraform HCL	Multi-cloud	Declarative infrastructure provisioning across cloud providers
CloudFormation	AWS	AWS-native infrastructure definition
ARM Templates	Azure	Azure resource deployment specifications
Kubernetes YAML	Container orchestration	Container workload definitions and configurations
Docker Compose	Container deployment	Multi-container application definitions
Ansible YAML	Configuration management	Server configuration and application deployment
Pulumi	Multi-cloud	Programmatic infrastructure using general-purpose languages

7.4 Tabular Specifications

Many deployment details are best captured in tabular form. Standard table formats include:

7.4.1 Node Specification Table

Table 10: Example Node Specification Table Format

Node ID	Type	vCPU	RAM	Storage	OS	Purpose
web-01	VM	4	16 GB	100 GB	Ubuntu 22.04	Web Server
app-01	VM	8	32 GB	200 GB	Ubuntu 22.04	App Server
db-01	VM	16	64 GB	1 TB	Ubuntu 22.04	Primary DB

7.4.2 Network Connection Table

Table 11: Example Network Connection Table Format

Source	Target	Protocol	Port(s)	Direction	Purpose
web-01	app-01	HTTPS	443	Outbound	API calls
app-01	db-01	PostgreSQL	5432	Outbound	Database queries
lb-01	web-*	HTTP/S	80, 443	Outbound	Load distribution

8 Viewpoint Metamodels

This section defines the conceptual metamodel underlying the Deployment Viewpoint. The metamodel establishes the vocabulary of element types, their properties, and valid relationships.

8.1 Core Metamodel

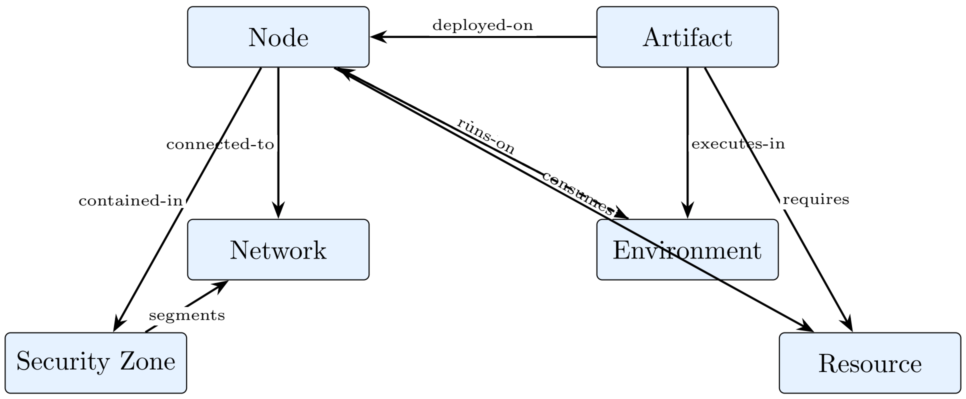


Figure 3: Deployment Viewpoint Core Metamodel

8.2 Entity Definitions

Entity: Node

Definition: A computational resource capable of hosting and executing software artifacts.

Attributes:

- `nodeId`: Unique identifier for the node
- `name`: Human-readable name
- `type`: Classification (physical, virtual, container, serverless)
- `hostname`: Network hostname
- `ipAddresses`: List of IP addresses
- `operatingSystem`: OS type and version
- `status`: Operational status (active, standby, maintenance)
- `location`: Physical or logical location
- `provider`: Infrastructure provider (on-premise, AWS, Azure, GCP)

Constraints:

- Each node must have a unique `nodeId`
- Virtual nodes must reference a parent physical node or hypervisor
- Container nodes must reference a container runtime environment

Entity: Artifact

Definition: A deployable unit of software that can be installed on and executed by a node.

Attributes:

- `artifactId`: Unique identifier
- `name`: Artifact name
- `version`: Version identifier
- `type`: Artifact type (executable, library, container image, configuration)
- `checksum`: Integrity verification hash
- `size`: Artifact size in bytes
- `repository`: Source repository location
- `buildInfo`: Build metadata (commit, timestamp, builder)

Constraints:

- Artifact version must follow semantic versioning
- Container images must include registry and tag information
- Configuration artifacts must specify target environment

Entity: Network

Definition: A communication infrastructure that connects nodes and enables data exchange.

Attributes:

- `networkId`: Unique identifier
- `name`: Network name
- `type`: Network type (LAN, WAN, VPC, overlay)
- `cidrBlock`: IP address range
- `vlanId`: VLAN identifier (if applicable)
- `mtu`: Maximum transmission unit
- `bandwidth`: Available bandwidth
- `latency`: Expected latency characteristics

Constraints:

- CIDR blocks must not overlap within the same network domain
- Connected nodes must have compatible network configurations

Entity: Environment

Definition: A runtime context that provides services and resources for artifact execution.

Attributes:

- `environmentId`: Unique identifier
- `name`: Environment name (e.g., production, staging, development)
- `type`: Environment type (bare metal, VM, container, serverless)
- `runtime`: Runtime platform (JVM, Node.js, Python, .NET)
- `runtimeVersion`: Runtime version
- `configuration`: Environment-specific configuration
- `variables`: Environment variables

Constraints:

- Environment configurations must be complete for artifact execution
- Sensitive configuration values must be stored securely

Entity: Security Zone

Definition: A logical boundary that groups nodes with similar security requirements and trust levels.

Attributes:

- `zoneId`: Unique identifier
- `name`: Zone name (e.g., DMZ, internal, restricted)
- `trustLevel`: Security classification level
- `accessPolicy`: Default access control policy
- `encryptionRequired`: Whether encryption is mandated
- `complianceRequirements`: Applicable compliance standards

Constraints:

- Cross-zone communication must traverse security controls
- Higher trust zones cannot directly access lower trust zones

Entity: Resource

Definition: A computational, storage, or network resource that can be allocated to nodes or artifacts.

Attributes:

- `resourceId`: Unique identifier
- `type`: Resource type (CPU, memory, storage, network bandwidth)
- `capacity`: Total available capacity
- `allocated`: Currently allocated amount
- `unit`: Unit of measurement
- `sharable`: Whether resource can be shared
- `elasticity`: Auto-scaling capability

Constraints:

- Allocated resources cannot exceed capacity
- Resource reservations must be honored by schedulers

8.3 Relationship Definitions

Table 12: Metamodel Relationship Definitions

Relationship	**Source**	**Target**	**Description**
deployed-on	Artifact	Node	Artifact is installed and runs on the node
connected-to	Node	Network	Node has network connectivity through this network
hosts	Node	Environment	Node provides the environment for execution
executes-in	Artifact	Environment	Artifact runs within this execution environment
contained-in	Node	Security Zone	Node is located within this security boundary
consumes	Node	Resource	Node uses this resource for operation
requires	Artifact	Resource	Artifact needs this resource to function
segments	Security Zone	Network	Zone defines a partition of the network
communicates-with	Node	Node	Direct communication path between nodes
replicates	Node	Node	Data or state replication relationship
fails-over-to	Node	Node	Failover target in case of primary failure
load-balances	Node	Node	Traffic distribution relationship

9 Conforming Notations

Several existing notations and modeling languages conform to the Deployment Viewpoint metamodel and may be used interchangeably or in combination.

9.1 UML 2.x Deployment Diagrams

The UML 2.x specification defines deployment diagrams as a standard notation for showing the physical arrangement of artifacts on nodes. UML deployment diagrams provide native support for nodes, artifacts, communication paths, and deployment specifications.

Conformance Level: Full conformance to core metamodel elements.

Tool Support: Enterprise Architect, Visual Paradigm, StarUML, PlantUML, Lucidchart, draw.io.

9.2 C4 Model - Deployment Diagram

The C4 model includes a deployment diagram type that shows how software systems and containers are deployed onto infrastructure. C4 deployment diagrams emphasize clarity and audience-appropriate abstraction levels.

Conformance Level: Partial conformance; focuses on container-level deployment rather than fine-grained artifacts.

Tool Support: Structurizr, PlantUML (C4 extension), IcePanel.

9.3 ArchiMate Technology Layer

ArchiMate's Technology Layer provides notation for modeling infrastructure elements including nodes, devices, system software, networks, and communication paths.

Conformance Level: Full conformance with extended enterprise architecture context.

Tool Support: Archi, BiZZdesign, MEGA, Sparx Enterprise Architect.

9.4 AWS/Azure/GCP Architecture Diagrams

Cloud providers offer official icon sets and diagram conventions for documenting cloud infrastructure deployments.

Conformance Level: Platform-specific conformance with rich service representation.

Tool Support: AWS Architecture Icons, Azure Architecture Icons, Google Cloud Architecture Diagramming Tool, Cloudcraft, Lucidchart, draw.io.

9.5 Kubernetes Architecture Diagrams

For containerized deployments, Kubernetes-specific notation captures clusters, nodes, pods, services, and networking constructs.

Conformance Level: Container orchestration-specific conformance.

Tool Support: Lens, k8sviz, KubeView, custom tooling.

Table 13: Notation Comparison Matrix

Feature	UML 2.x	C4 Model	ArchiMate	Cloud Icons	K8s Native	Custom
Physical Nodes	●	●	●	●	○	●
Virtual Machines	●	●	●	●	○	●
Containers	○	●	○	●	●	●
Networks	●	○	●	●	●	●
Security Zones	○	○	●	●	○	●
Cloud Services	○	○	○	●	○	●
Standardized	●	●	●	○	○	–
Tool Support	●	●	●	●	○	○

● = Strong support, ○ = Limited support, – = Not applicable

10 Model Correspondence Rules

Model correspondence rules define how elements in deployment models relate to elements in other architectural views. These rules ensure consistency across the architecture documentation.

10.1 Component-and-Connector View Correspondence

Correspondence Rule CR-01: Component to Artifact Mapping

Rule: Every runtime component identified in a Component-and-Connector view must correspond to one or more deployment artifacts.

Formal Expression:

$$\forall c \in Components_{C\&C} : \exists a \in Artifacts_{Deploy} : manifests(a, c)$$

Rationale: Ensures that all software components have a concrete deployment representation.

Verification: Traceability matrix linking components to artifacts.

Correspondence Rule CR-02: Connector to Network Path Mapping

Rule: Every connector between components in a C&C view that crosses node boundaries must have a corresponding network path in the deployment model.

Formal Expression:

$$\forall conn(c_1, c_2) : node(c_1) \neq node(c_2) \Rightarrow \exists path(node(c_1), node(c_2))$$

Rationale: Validates that required communication paths exist in the infrastructure.

Verification: Network connectivity analysis.

10.2 Module View Correspondence

Correspondence Rule CR-03: Module to Artifact Packaging

Rule: The module decomposition structure should be reflected in artifact packaging boundaries.
Formal Expression:

$$\forall m \in Modules : packaged\_in(m) \subseteq Artifacts$$

Rationale: Maintains alignment between logical module structure and physical deployment units.
Verification: Build manifest analysis.

10.3 Data View Correspondence

Correspondence Rule CR-04: Data Store to Storage Node Mapping

Rule: Every persistent data store identified in a data architecture view must be allocated to storage infrastructure in the deployment model.
Formal Expression:

$$\forall ds \in DataStores : \exists n \in Nodes : hosts(n, ds) \wedge type(n) = storage$$

Rationale: Ensures all data has defined storage infrastructure.
Verification: Data residency mapping.

10.4 Security View Correspondence

Correspondence Rule CR-05: Trust Boundary to Security Zone Mapping

Rule: Trust boundaries defined in security architecture must align with security zone definitions in the deployment model.
Formal Expression:

$$\forall tb \in TrustBoundaries : \exists sz \in SecurityZones : implements(sz, tb)$$

Rationale: Validates that security architecture is realized in deployment.
Verification: Security zone audit.

10.5 Correspondence Verification Matrix

Table 14: Cross-View Correspondence Verification

Rule ID	Source View	Target Elements	Verification Method
CR-01	C&C Components	Deployment Artifacts	Traceability matrix
CR-02	C&C Connectors	Network Paths	Connectivity analysis
CR-03	Module Structure	Artifact Packages	Build manifest review
CR-04	Data Stores	Storage Nodes	Data residency mapping
CR-05	Trust Boundaries	Security Zones	Security zone audit

11 Operations on Views

This section defines the methods and procedures for creating, interpreting, analyzing, and implementing deployment views.

11.1 Creation Methods

11.1.1 View Development Process

Step 1: Establish Scope and Context

1. Identify the system or subsystem to be documented
2. Determine the deployment scenarios (development, staging, production)
3. Identify target infrastructure platforms
4. Gather existing infrastructure documentation
5. Interview key stakeholders (operations, security, development)

Step 2: Identify Deployment Elements

1. List all deployable software artifacts from Component-and-Connector views
2. Identify required runtime environments and dependencies
3. Document hardware and virtual infrastructure nodes
4. Map network topology and connectivity requirements
5. Define security zones and trust boundaries

Step 3: Create Initial Deployment Model

1. Draw high-level deployment topology showing major nodes
2. Allocate software artifacts to execution nodes
3. Define network connections and communication paths
4. Overlay security zone boundaries
5. Document resource allocations and constraints

Step 4: Elaborate and Refine

1. Add detailed node specifications (hardware, OS, middleware)
2. Document environment-specific configurations
3. Specify high availability and redundancy mechanisms
4. Define scaling policies and capacity limits
5. Add monitoring and observability infrastructure

Step 5: Validate and Review

1. Verify correspondence with other architectural views
2. Validate against quality attribute requirements
3. Review with infrastructure and security teams
4. Conduct deployment feasibility assessment
5. Document decisions, rationale, and trade-offs

11.1.2 Documentation Templates

Node Documentation Template:

```
Node Specification
==================
Node ID:        [Unique identifier]
Name:           [Human-readable name]
Type:           [Physical | Virtual | Container | Serverless]
Purpose:        [Primary function and role]

Hardware/Resources:
  - CPU:        [Cores/vCPUs and type]
  - Memory:     [RAM capacity]
  - Storage:    [Disk capacity and type]
  - Network:    [Network interfaces]

Software Stack:
  - OS:         [Operating system and version]
  - Runtime:    [Runtime environments]
  - Middleware: [Middleware components]

Network Configuration:
  - Hostname:   [FQDN]
  - IP Address: [Static IP or DHCP]
  - DNS:        [DNS entries]
  - Ports:      [Open ports and protocols]

Deployed Artifacts:
  - [List of deployed software components]

Dependencies:
  - [External service dependencies]

Notes:
  - [Additional configuration notes]
```

Listing 1: Node Specification Template

11.1.3 Common Patterns and Idioms

Table 15: Deployment Patterns

Pattern	Description	Use When
Single Server	All components on one node	Development, small-scale applications
N-Tier	Separate tiers for web, app, data	Traditional enterprise applications
Microservices	Independent service deployment	Scalable, independently deployable services
Container Cluster	Orchestrated container deployment	Cloud-native, dynamic scaling needs
Serverless	Function-based deployment	Event-driven, variable workloads
Hybrid Cloud	Mix of on-premise and cloud	Data sovereignty, burst capacity
Multi-Region	Geographic distribution	High availability, low latency globally
Blue-Green	Parallel production environments	Zero-downtime deployments
Canary	Gradual traffic shifting	Risk mitigation for new releases

11.2 Interpretive Methods

11.2.1 Reading Deployment Diagrams

When interpreting a deployment view, stakeholders should follow this systematic approach:

1. **Identify the Scope:** Determine what system or subsystem is depicted and for which environment(s).
2. **Understand Node Types:** Distinguish between physical servers, virtual machines, containers, and cloud services based on notation and stereotypes.
3. **Trace Software Allocation:** For each software component of interest, identify which node(s) host it and in what configuration.
4. **Analyze Connectivity:** Follow communication paths between nodes, noting protocols, ports, and any intermediate network devices.
5. **Identify Security Boundaries:** Locate security zone boundaries and understand the trust levels and access controls between zones.
6. **Assess Redundancy:** Look for redundant nodes, load balancers, and failover paths that support availability requirements.
7. **Check Resource Allocations:** Review resource specifications to understand capacity and potential constraints.

11.2.2 Stakeholder-Specific Interpretation Guides

For System Administrators: Focus on node specifications, installed software, network configurations, and operational procedures. Pay attention to monitoring integration points and log aggregation paths.

For Security Teams: Prioritize security zone boundaries, network segmentation, encryption points, and access control mechanisms. Validate that sensitive components are appropriately isolated.

For Development Teams: Understand the runtime environments available, deployed artifact versions, configuration management approaches, and paths from development to production.

For Capacity Planners: Examine resource allocations, scaling policies, current utilization levels, and growth projections. Identify potential bottlenecks and scaling limits.

11.3 Analysis Methods

11.3.1 Performance Analysis

Performance Analysis Technique

Purpose: Evaluate whether the deployment topology can meet performance requirements.

Inputs:

- Deployment model with resource specifications
- Performance requirements (response time, throughput)
- Workload characteristics (request rates, data volumes)

Process:

1. Model system as a queuing network
2. Assign service rates based on resource specifications
3. Apply workload arrival rates
4. Calculate utilization, queue lengths, and response times
5. Compare against requirements

Outputs:

- Predicted performance metrics
- Identified bottlenecks
- Scaling recommendations

11.3.2 Availability Analysis

Availability Analysis Technique

Purpose: Calculate expected system availability based on deployment redundancy.

Inputs:

- Deployment topology with redundancy configuration
- Component reliability data (MTBF, MTTR)
- Failover mechanisms and recovery times

Process:

1. Construct reliability block diagram from deployment
2. Identify serial and parallel configurations
3. Calculate composite availability using formulas:
    - Serial: $A_{total} = A_1 \times A_2 \times ... \times A_n$
    - Parallel: $A_{total} = 1 - (1 - A_1) \times (1 - A_2) \times ... \times (1 - A_n)$
4. Account for failover time impacts
5. Compare against availability targets

Outputs:

- Predicted availability percentage
- Single points of failure identification
- Redundancy improvement recommendations

11.3.3 Security Analysis

Security Posture Analysis Technique

Purpose: Evaluate the security effectiveness of the deployment architecture.

Inputs:

- Deployment model with security zones
- Network segmentation configuration
- Access control policies
- Threat model and attack vectors

Process:

1. Map attack surface from external entry points
2. Trace potential attack paths through the deployment
3. Evaluate defense-in-depth layers
4. Assess data protection at rest and in transit
5. Review compliance against security requirements

Outputs:

- Security risk assessment
- Vulnerability identification
- Hardening recommendations

11.3.4 Cost Analysis

Total Cost of Ownership Analysis

Purpose: Estimate the total cost of the deployment configuration.

Inputs:

- Deployment model with all resources
- Pricing information (hardware, cloud, licensing)
- Operational cost factors
- Time horizon for analysis

Process:

1. Calculate infrastructure costs (compute, storage, network)
2. Add software licensing costs
3. Estimate operational costs (personnel, support)
4. Project costs over analysis period
5. Compare alternatives if applicable

Outputs:

- Total cost estimate
- Cost breakdown by category
- Cost optimization opportunities

11.4 Implementation Methods

11.4.1 Deployment Automation

Modern deployment implementation relies heavily on automation. The deployment view serves as the specification for Infrastructure as Code implementations.

Table 16: Implementation Automation Approaches

Approach	Tools	Description
Infrastructure Provisioning	Terraform, CloudFormation, Pulumi	Declaratively provision cloud and on-premise infrastructure
Configuration Management	Ansible, Chef, Puppet, Salt	Configure operating systems and install software
Container Orchestration	Kubernetes, Docker Swarm, Nomad	Deploy and manage containerized workloads
CI/CD Pipelines	Jenkins, GitLab CI, GitHub Actions	Automate build, test, and deployment workflows
GitOps	ArgoCD, Flux	Git-driven declarative infrastructure management

11.4.2 Deployment Verification

After implementation, verify that the actual deployment matches the documented architecture:

1. **Infrastructure Validation:** Verify that all specified nodes exist with correct configurations using infrastructure scanning tools.
2. **Artifact Verification:** Confirm that correct versions of software artifacts are deployed to the appropriate nodes.
3. **Connectivity Testing:** Validate network paths between nodes using network diagnostic tools.
4. **Security Verification:** Audit security zone configurations and access controls against specifications.
5. **Performance Baseline:** Establish baseline performance metrics for comparison against requirements.

12 Examples

This section provides concrete examples of deployment views for common scenarios.

12.1 Example 1: Three-Tier Web Application

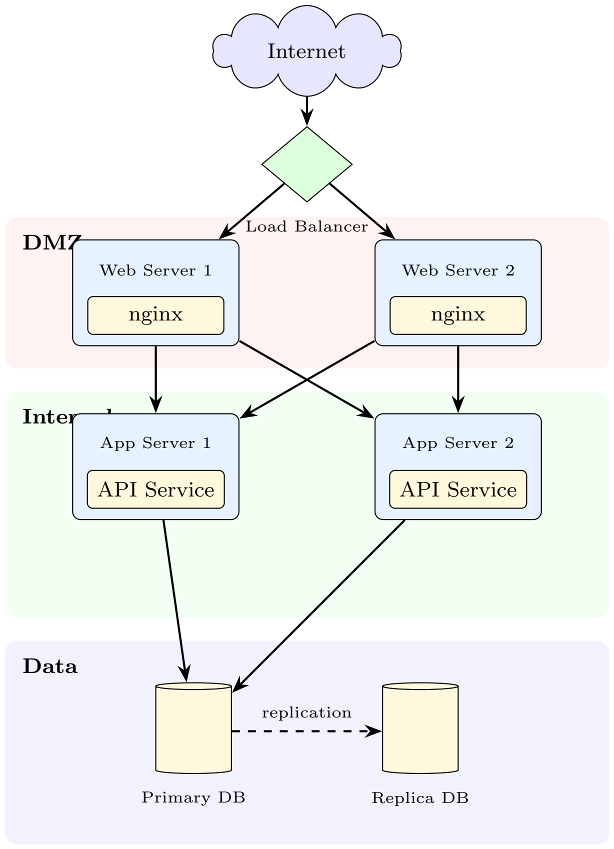


Figure 4: Three-Tier Web Application Deployment

Description: This example shows a classic three-tier architecture deployed across three security zones. The DMZ hosts web servers accessible from the internet through a load balancer. Application servers reside in the internal zone, and databases are isolated in a dedicated data zone with primary-replica replication.

12.2 Example 2: Kubernetes Microservices Deployment

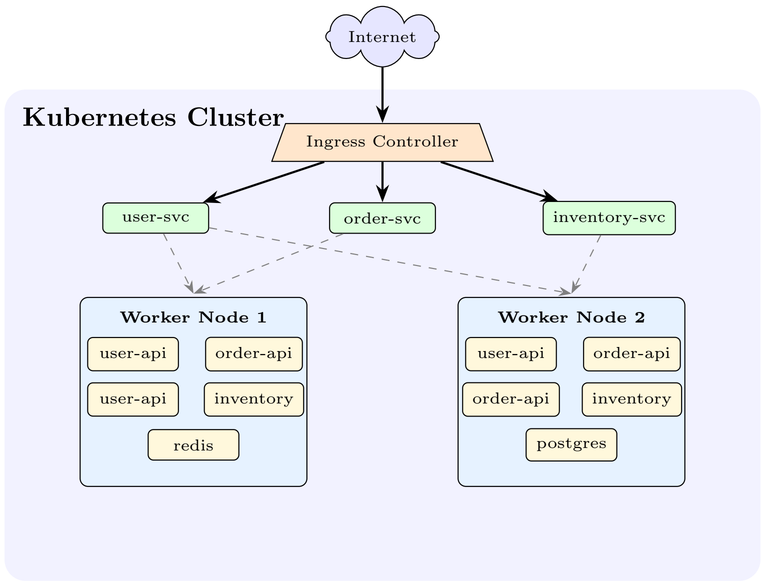


Figure 5: Kubernetes Microservices Deployment

Description: This example depicts a microservices architecture deployed on Kubernetes. An ingress controller routes external traffic to internal services. Pods are distributed across worker nodes for high availability, with services providing stable network endpoints for pod groups.

12.3 Example 3: Multi-Region Cloud Deployment

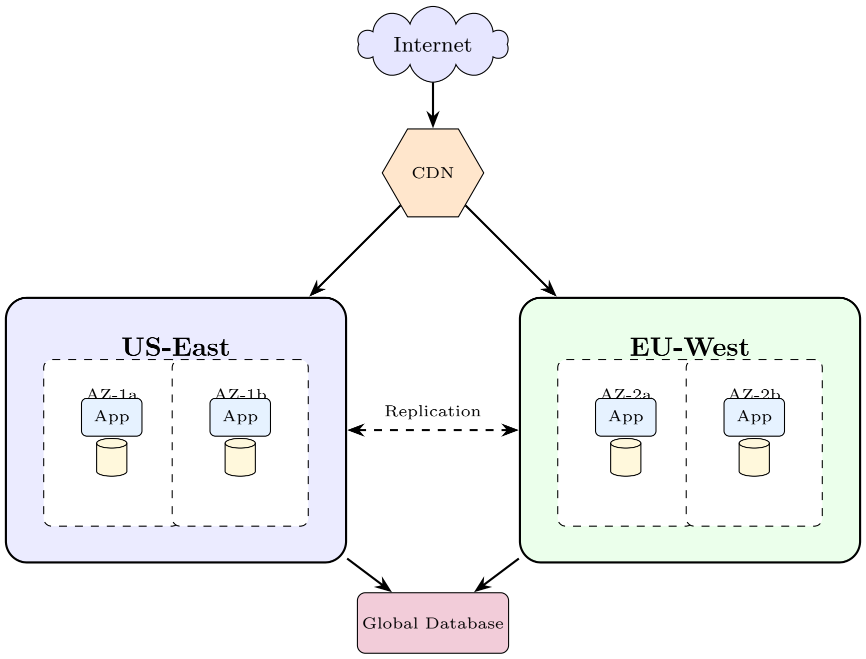


Figure 6: Multi-Region Cloud Deployment with Global Database

Description: This example shows a globally distributed deployment across two cloud regions (US-East and EU-West). Each region contains multiple availability zones for high availability. A CDN provides edge caching, and a global database service handles cross-region data consistency.

13 Notes

13.1 Versioning Considerations

Deployment views should be versioned alongside the systems they document. Key versioning practices include:

- Maintain separate deployment views for each environment (development, staging, production)
- Version deployment documentation with the software release it describes
- Document deltas between versions when significant changes occur
- Archive historical deployment configurations for audit and rollback purposes

13.2 Tooling Recommendations

Table 17: Recommended Tools by Use Case

Use Case	**Recommended Tools**
Diagram Creation	draw.io, Lucidchart, Visio, PlantUML, Structurizr
Infrastructure as Code	Terraform, Pulumi, AWS CDK, Azure Bicep
Documentation	Confluence, GitBook, Notion, Markdown/LaTeX
Architecture Modeling	Archi, Enterprise Architect, Structurizr
Cloud Visualization	Cloudcraft, Hava, Hyperglance, AWS Architecture Icons
Container Visualization	Lens, Octant, k9s, KubeView

13.3 Common Pitfalls

Common Mistakes to Avoid

1. **Outdated Documentation:** Deployment views that don't reflect current infrastructure lose value quickly
2. **Excessive Detail:** Including too much low-level detail obscures important architectural decisions
3. **Missing Security Context:** Omitting security zones and trust boundaries leaves security gaps
4. **Single Environment Focus:** Documenting only production ignores important environment differences
5. **No Correspondence Validation:** Failing to verify alignment with other architectural views
6. **Static View Only:** Not documenting scaling, failover, and operational behavior

13.4 Evolution and Maintenance

Deployment documentation should be treated as a living artifact that evolves with the system:

- Establish ownership and review cycles for deployment documentation
- Integrate deployment view updates into change management processes
- Automate documentation generation where possible using IaC as source of truth
- Conduct periodic reviews to ensure accuracy and relevance
- Use deployment documentation in incident post-mortems and capacity planning

14 Sources

14.1 Primary References

1. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., & Stafford, J. (2010). *Documenting Software Architectures: Views and Beyond* (2nd ed.). Addison-Wesley Professional.
2. ISO/IEC/IEEE 42010:2011. *Systems and software engineering — Architecture description*. International Organization for Standardization.
3. Object Management Group. (2017). *Unified Modeling Language Specification Version 2.5.1*. OMG Document formal/2017-12-05.
4. Brown, S. (2018). *The C4 Model for Visualising Software Architecture*. Leanpub.
5. The Open Group. (2022). *ArchiMate 3.2 Specification*. The Open Group.

14.2 Supplementary References

6. Rozanski, N., & Woods, E. (2012). *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives* (2nd ed.). Addison-Wesley Professional.
7. Morris, K. (2020). *Infrastructure as Code: Dynamic Systems for the Cloud Age* (2nd ed.). O'Reilly Media.
8. Burns, B., Beda, J., & Hightower, K. (2019). *Kubernetes: Up and Running* (2nd ed.). O'Reilly Media.
9. Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.
10. Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional.

14.3 Online Resources

- AWS Architecture Center: `https://aws.amazon.com/architecture/`
- Azure Architecture Center: `https://docs.microsoft.com/en-us/azure/architecture/`
- Google Cloud Architecture Framework: `https://cloud.google.com/architecture/framework`
- C4 Model: `https://c4model.com/`
- Kubernetes Documentation: `https://kubernetes.io/docs/`

A Deployment View Checklist

Use this checklist to verify completeness of deployment documentation:

Item	Complete?
Scope and Context	
System/subsystem clearly identified	☐
Target environments documented	☐
Stakeholders identified	☐
Infrastructure Elements	
All nodes documented with specifications	☐
Network topology defined	☐
Security zones established	☐
Storage infrastructure specified	☐
Software Allocation	
All artifacts mapped to nodes	☐
Runtime environments specified	☐
Dependencies documented	☐
Configurations defined per environment	☐
Quality Attributes	
High availability mechanisms documented	☐
Scaling approach defined	☐
Security controls specified	☐
Performance considerations addressed	☐
Operations	
Deployment procedures documented	☐
Monitoring integration specified	☐
Backup and recovery defined	☐
Operational runbooks available	☐
Validation	
Correspondence with C&C view verified	☐
Stakeholder review completed	☐
Implementation matches documentation	☐

B Glossary

Artifact	A deployable unit of software such as an executable, library, container image, or configuration file.
Availability Zone	A physically separate location within a cloud region, providing isolation from failures in other zones.
Container	A lightweight, standalone executable package that includes everything needed to run software.
Deployment	The process of making software available for use on target infrastructure.

Environment	A complete set of infrastructure and configuration for running software (e.g., development, staging, production).
Failover	The automatic switching to a redundant system upon failure of the primary system.
High Availability	A characteristic of systems designed to be operational for a high percentage of time.
Infrastructure as Code	The practice of managing infrastructure through machine-readable definition files.
Load Balancer	A device or service that distributes network traffic across multiple servers.
Node	A computational resource capable of hosting and executing software.
Pod	The smallest deployable unit in Kubernetes, consisting of one or more containers.
Region	A geographic area containing one or more data centers operated by a cloud provider.
Replication	The process of copying data between nodes to ensure consistency and availability.
Security Zone	A network segment with defined security policies and trust levels.
Scaling	The ability to increase or decrease resources to handle varying workloads.
Virtual Machine	A software emulation of a physical computer that runs an operating system.