Architectural Blueprint: A Zero-Trust Security Framework for Cloud-Native Workloads via SDN

Part 1: Foundational Architecture: Controller, Environment, and Availability

This section establishes the core components of the Software-Defined Networking (SDN) control plane, justifying the selection of the controller, designing its high-availability architecture, and defining the persistent state model necessary for a production-grade system.

1.1 Controller Framework Selection: Justification for Ryu

The project's requirements necessitate an industry-standard, Python-based SDN controller framework that offers the flexibility to implement custom security logic, machine learning (ML) integration, and a bespoke high-availability (HA) model.[1]

The primary open-source candidates for programmable network control are Ryu, ONOS, and Floodlight.

- **Ryu:** A component-based, Python-native framework.[2] Its architecture is often described as a "toolbox" [4], providing robust libraries for OpenFlow (supporting versions 1.0 through 1.5) [3], packet processing [4], and event handling. This design leaves the application logic entirely to the developer, making it the ideal choice for a sophisticated, custom Python application that must integrate ML [7] and Kubernetes-specific logic. Performance benchmarks are highly competitive, with studies frequently showing Ryu exhibits the lowest latency, jitter, and packet loss compared to its peers, making it suitable for security-sensitive applications.[9]

- **ONOS & OpenDaylight (ODL):** These are more "platform-centric," Java-based controllers.[3] Their primary advantages are robust, *built-in* clustering mechanisms for high availability [1] and more extensive, pre-built Northbound API (NBI) libraries.[1] However, their Java core makes deep, native Python-based ML integration and custom application development (as required by this project) more complex. Python is typically relegated to a client of their REST APIs rather than a core component of the controller logic.

Architectural Decision: Ryu is the selected framework. The requirement for a sophisticated *Python application* with ML and Kubernetes integration decisively favors Ryu's Python-native, modular, and unopinionated architecture.

While research indicates that Ryu, unlike ONOS, has no *inbuilt* clustering mechanism [1], this is an architectural advantage for this project's goals. Relying on an opaque, internal clustering mechanism (like that of ONOS) creates a "black box." Ryu's model, by contrast, *compels* the use of external, industry-standard coordination tools like Zookeeper.[4] This allows for the implementation of a well-understood, transparent, and battle-tested Master/Slave leader election pattern.[18] This design is inherently more modular, observable, and aligned with a microservice-based, "production-ready" ethos, as the system is not locked into a monolithic controller's specific HA implementation.

Table 1: Comparative Analysis of SDN Controller Frameworks

Feature	Ryu	ONOS	Floodlight
Primary Language	Python [2]	Java [3]	Java [3]
Performance	Excellent (low latency, jitter, packet loss) [9]	High Throughput [14]	High Packet Loss / Latency [9]
HA/Clustering Model	External tools (e.g., Zookeeper) [1]	Built-in, platform-native [1]	External tools [1]
Northbound API	Limited REST API (build your own) [1]	Extensive REST, gRPC [1]	REST API
Python/ML Suitability	**Excellent.** Python-native "toolbox" [4], ideal	**Poor.** Requires Python to act as a client to the Java	**Poor.** Java-based.

	for custom ML.[7]	platform.	

1.2 High Availability (HA) Architecture for a Ryu Cluster

To meet the "fault tolerance" requirement and eliminate the controller as a single point of failure [21], a multi-controller architecture is essential. This design uses an external coordinator for leader election, creating a resilient Master-Slave model.[17]

1. **Zookeeper as Coordinator:** A Zookeeper ensemble (e.g., a 3-node cluster) will be deployed as the external coordination service.[4] Zookeeper is the industry standard for this use case, providing reliable distributed synchronization, configuration management, and leader election.[24]
2. **Controller Deployment:** Multiple instances of the Ryu application will be deployed (e.g., as Kubernetes Pods or Docker containers). All instances will be configured with the connection string for the Zookeeper ensemble.
3. **Python Implementation (Leader Election):** The Python kazoo library, a high-level Zookeeper client, will be used to implement leader election.[28]
   - Each Ryu instance will instantiate kazoo.recipe.election.Election on a shared "election" znode path.[28]
   - This recipe transparently implements the standard Zookeeper algorithm: all instances attempt to create an *ephemeral, sequential* znode.[20]
   - The instance that successfully creates the znode with the *lowest sequence number* wins the election.[35]
   - Crucially, each non-leader (slave) instance *watches* only the znode directly preceding it in the sequence.[20] This efficient approach avoids the "herd effect," where all slaves would be notified (and contend) simultaneously upon leader failure.
4. **Python Implementation (HA Logic):**
   - The election.run(my_leader_function) method [28] will be executed in a dedicated eventlet green thread, which is compatible with Ryu's asynchronous model.[37]
   - **When Election is Won:** The my_leader_function is called. This instance is now the **Master**. Its primary task is to iterate through all connected OpenFlow switches (datapaths) and send an OpenFlow OFPRoleRequest message, setting its role to OFPCR_ROLE_MASTER.[17] This OpenFlow-native command instructs the switches to send all asynchronous messages (like Packet-In) to this controller and to accept its Flow-Mod (flow rule) commands.
   - **When Election is Lost:** If the Zookeeper session is lost, the leader's ephemeral znode is automatically deleted.[40] This causes the election.run() function to exit. The instance reverts to a **Slave** role and should ideally send an OFPRoleRequest with

role=OFPCR_ROLE_SLAVE.[38]

    - **Slave State:** All non-leader instances will use kazoo.recipe.watchers.ChildrenWatch [42] to monitor the election znode and will re-enter the election process as soon as their preceding znode disappears.

5. **Data-Plane Configuration:** All Open vSwitch (OVS) instances in the data plane will be configured to connect to *all* Ryu controller instances simultaneously (e.g., ovs-vsctl set-controller br0 tcp:c1_ip:6653 tcp:c2_ip:6653...).[45] The OpenFlow multi-controller protocol, managed via the OFPRoleRequest message, ensures that while many controllers are *connected*, only the one with the MASTER role is actively managing the switch.[47]

1.3 Database Persistence for State and Policy

A production-ready system requires persistence of network state and policies.[48] A critical design decision is the separation of coordination (ephemeral state) from policy persistence (durable state).

- **Zookeeper vs. PostgreSQL:** Zookeeper is a *Coordination Service*, not a database.[26] It is purpose-built for ephemeral data, leader election, and service discovery.[24] It is unsuitable for storing the durable "source of truth" for network policies. For this, a relational database like **PostgreSQL** will be used, which integrates well with modern Python API frameworks.[49]

This separation enables a highly robust and decoupled architecture for state synchronization, which is essential for high availability. In a multi-controller setup, state must be synchronized.[16] A naive approach would involve complex, direct controller-to-controller communication. A far superior, production-grade architecture is to *externalize the source of truth*. The API and the Controller will not communicate directly; they will communicate *via* the database.

This architectural flow is as follows:

1. A user or system (like the ML module) submits a declarative "intent" (a security policy) to the Northbound REST API (defined in Part 3).
2. The FastAPI service (Part 3) validates this intent and *writes* it as a row in a policies table in the PostgreSQL database.[50]
3. The **Master Ryu Controller** (elected in section 1.2) is the *only* component that reads from this policies table. It polls or uses a database notification mechanism (e.g., PostgreSQL's LISTEN/NOTIFY) to detect changes.
4. When it detects a new or updated policy, it translates this abstract intent into concrete

OpenFlow rules and installs them on the switches.

This design's resilience is demonstrated in a failover scenario:

1. The Master Ryu controller (ryu-1) crashes or is disconnected.
2. Zookeeper detects the failure (its ephemeral znode disappears).[41] A new leader election is triggered.[20]
3. A Slave controller (ryu-2) wins the election and becomes the new Master.
4. ryu-2's *first action* as Master is to (a) send OFPRoleRequest messages [39] and (b) connect to the PostgreSQL DB and read the *entire* set of policies from the policies table.
5. It then reconciles the network state, re-installing all flow rules necessary to match the "source of truth" defined in the database.

State synchronization is thus achieved *without* any direct, fragile controller-to-controller communication. The database serves as the source of truth for *intent*, aligning perfectly with Intent-Based Networking principles.[48]

Part 2: The Core Application: Dynamic Policy Enforcement in Kubernetes

This section details the primary application logic: implementing a Zero-Trust security model by dynamically programming the network fabric in response to events from a Kubernetes (K8s) cluster.

2.1 Defining the Zero-Trust Environment (NIST SP 800-207)

The core problem domain is the implementation of a Zero-Trust Architecture (ZTA), as defined by NIST SP 800-207.[53] This model translates theoretical principles into a concrete, enforceable network architecture.

- "Never Trust, Always Verify" [53]: The foundational tenet. The network will not implicitly trust any entity, regardless of its location (e.s., inside or outside the perimeter). All connectivity must be explicitly authorized.
- "Least Privilege Access" [53]: Policies will be granular, permitting *only* the specific ports and protocols required for a microservice to perform its function, and nothing more.
- Microsegmentation [58]: This is the *core mechanism* for achieving ZTA. Instead of broad,

perimeter-based segments (like VLANs) [61], we will use SDN to create granular, software-defined segments around individual workloads (K8s Pods). The centralized control and programmability of SDN are what make *dynamic* microsegmentation possible.[59]

- Dynamic and Contextual Policy [61]: This is the key capability. A true ZTA policy moves *beyond* static IP addresses.[61] The policy enforcement must be based on "context" or "attributes".[61] In our architecture, these attributes will be **Kubernetes labels** (e.g., app=frontend, env=production). A policy will be defined as "app=frontend can talk to app=backend," not as "10.1.1.5 can talk to 10.1.1.6."

2.2 The "Priority Override" Architecture for CNI Co-existence

A significant real-world challenge is integrating a custom SDN controller with an existing Kubernetes cluster, which already has its own networking solution. Modern K8s clusters often use a Container Network Interface (CNI) plugin like OVN-Kubernetes, which itself is built on Open vSwitch (OVS) and OpenFlow.[64]

The challenge is a *controller conflict*. OVN-Kubernetes (and similar CNIs) works by running its *own* local controller (ovn-controller) on each K8s node.[66] This ovn-controller connects to the local OVS bridge (br-int) and actively programs OpenFlow rules to provide Pod-to-Pod connectivity, K8s Service load balancing, and overlay tunnels.[67] We cannot simply *replace* this CNI controller, as it handles the complex and essential baseline networking.[73]

The solution is a **Security Overlay** architecture based on OpenFlow's inherent features:

1. **Multi-Controller OVS:** The OVS bridge on each K8s node will be configured to connect to *both* controllers: the local ovn-controller (typically via a Unix domain socket) and our *external* Ryu controller cluster (via TCP).[45]
2. OpenFlow Priority [2]: This is the core mechanism. OpenFlow flow tables are processed by priority. When a packet enters a switch, it is matched against the *highest-priority* rule that it matches.[2]
3. **CNI Controller (OVN-K8s):** Installs *low-priority* (e.g., priority=100) rules that provide the *baseline cluster connectivity*.[80] This can be conceptualized as the "default allow" policy within the cluster.
4. **Ryu Controller (ZTA PEP):** Our application will *only* install *high-priority* (e.g., priority=5000) rules that enforce our Zero-Trust policies.[81]

This "priority override" model creates a non-intrusive ZTA solution. While ZTA purists advocate a "default-deny" model [53], this is impractical in a K8s cluster where the CNI's job is to provide "default-allow" connectivity.[65] This architecture achieves the *same security outcome* by layering high-priority *deny* (or explicit allow) rules over the CNI's low-priority *allow* rules.

Consider the following packet-flow scenario:

- **Goal:** A ZTA policy is active: DENY traffic from app=frontend to app=db on TCP port 5432.
- **OVN-K8s CNI Rule (in OVS):** priority=100, actions=output:NORMAL (A simplified rule representing the CNI's baseline "allow" logic [70]).
- **Our Ryu App Rule (in OVS):** priority=5000, ip, tcp, tcp_dst=5432, [match_frontend_pods], [match_db_pods], actions=DROP.[81]

Now, two packets arrive at the OVS bridge:

1. **Packet 1 (Malicious):** A packet from a frontend Pod is sent to a db Pod on port 5432.
   - The packet enters the OVS pipeline.
   - It matches *both* the CNI's rule (priority 100) and our Ryu app's rule (priority 5000).
   - The OVS switch executes *only* the highest-priority matching rule.[77]
   - **Result:** The packet matches the priority=5000 rule and is **DROPPED**.
2. **Packet 2 (Legitimate):** A packet from a frontend Pod is sent to an api Pod on port 8080.
   - The packet enters the OVS pipeline.
   - It *does not* match our specific priority=5000 rule.
   - It *only* matches the CNI's priority=100 rule.
   - **Result:** The packet is matched and **ALLOWED** by the CNI, as intended.

This "priority override" architecture is the central, non-intrusive mechanism for integrating our ZTA framework with any OpenFlow-based CNI. It allows the CNI to handle baseline connectivity while our application layers its security policy on top.

2.3 The Policy Decision Point (PDP): A Kubernetes-Aware Controller

The heart of the Ryu application is the Policy Decision Point (PDP), which translates abstract *intent* (K8s labels) into concrete *state* (OpenFlow rules).

- **Component: Kubernetes API Client:** The Ryu application will utilize the official kubernetes Python client library.[83]
  - **Authentication:** When the Ryu controller is deployed as a Pod *inside* the K8s cluster, it will use config.load_incluster_config().[83] This function seamlessly uses the Pod's mounted ServiceAccount token [90] to authenticate securely to the K8s API server. For local development, config.load_kube_config() will be used to load credentials from the local kubeconfig file.[84]

- **Event-Driven Reconciliation Logic:** The controller will be event-driven, not poll-based.
  1. **Internal State:** The Master controller will maintain two in-memory dictionaries:
     - pod_label_map: Maps Pod IP addresses to their corresponding K8s labels.
     - policy_map: Stores the active security policies read from the PostgreSQL database (as per section 1.3).
  2. **K8s Event Watchers:** The controller will run two watch streams [84] in background green threads:
     - v1.list_namespaced_pod(watch=True): This provides a real-time stream of all Pod events (ADDED, MODIFIED, DELETED).
     - A watcher for policy changes. This could be v1.list_network_policy(watch=True) [91] if using native K8s NetworkPolicy objects, or more robustly, a listener on the PostgreSQL DB (as per 1.3).
  3. **Reconciliation Loop:** This logic is triggered by events.
     - **On Pod Event (e.g., "Pod ADDED" with IP 10.1.1.7 and labels {'app': 'db'}):**
       1. The controller updates its internal pod_label_map.
       2. It triggers a "policy reconciliation" for this new Pod.
       3. It iterates through the policy_map to find all policies that *match* this Pod's new labels (e.g., any policy with a source or destination selector for app=db).[92]
       4. For each matching policy, it (re)calculates the full set of source/destination Pod IPs and installs the necessary high-priority OpenFlow rules [80] to all OVS bridges.
     - **On Policy Event (e.g., "Policy ADDED: DENY app=frontend -> app=db"):**
       1. The controller adds the policy to its policy_map.
       2. It finds all Pod IPs that match the source (app=frontend) and destination (app=db) label selectors by querying its pod_label_map.[94]
       3. It installs a new priority=5000, ip, ip_src=[frontend_ips], ip_dst=[db_ips], actions=DROP flow rule to all switches.

Ryu's framework is event-driven (e.g., EventOFPPacketIn) [6], as is the K8s client watch.[84] A clean, modular design must bridge these two event loops. Instead of polluting the K8s watcher thread with OpenFlow logic, a superior design will be used:

1. Define *custom Ryu events* (e.g., EventK8sPodUpdate(pod_obj) and EventK8sPolicyUpdate(policy_obj)).
2. The background thread running the K8s watch [84] will do *only one thing*: when it receives an event from the K8s API, it will emit the corresponding custom Ryu event (e.g., self.send_event_to_observers(EventK8sPodUpdate(pod))).
3. A *separate* Ryu application (e.g., PolicyEnforcementApp) will register handlers for these custom events (e.g., @set_ev_cls(EventK8sPodUpdate)).
4. The policy reconciliation and flow rule installation logic will reside *inside these handlers*, which run safely within Ryu's main event loop.

This architecture cleanly decouples the *data gathering* (the K8s watcher) from the *network-facing action* (the policy enforcer), representing a robust and modular software design.

Part 3: The Northbound Interface: An Intent-Based Networking (IBN) API

This section defines the external automation interface, abstracting the system's complexity by providing a declarative, Intent-Based Networking (IBN) API for policy management.[95]

3.1 Principles of Intent-Based Networking (IBN) Implementation

The project requires the implementation of IBN principles.[52] This represents an evolution from imperative SDN.

- **SDN API (Imperative):** An imperative API exposes the "how." For example, POST /switches/1/flows with a body describing a specific OpenFlow rule.
- **IBN API (Declarative):** A declarative API exposes the "what." For example, POST /policies with a body describing a *business goal*, such as "Isolate production workloads from staging".[52]

The proposed architecture implements a true IBN system:

1. **Translation (Intent -> Policy):** The user submits a high-level JSON object (the "intent") to the API.[52]
2. **Automated Implementation:** The API service persists this intent in the PostgreSQL database (as per 1.3). The Ryu controller (as per 2.3) independently detects this intent, *translates* it (using K8s labels) into concrete network configurations (OpenFlow rules), and *implements* them on the data plane.
3. **Continuous Monitoring & Adaptation:** The controller *continuously monitors* the K8s API (as per 2.3) for "drift" (e.g., new Pods scaling up). It *automatically adapts* the network configuration to keep it compliant with the original *intent*.[52] This "closed-loop validation" [97] is the essence of IBN.

3.2 API Framework Design: FastAPI

A high-performance Python framework is required for the REST API. While frameworks like Flask are popular [49], **FastAPI** is the clear architectural choice.

- **Performance:** FastAPI is a modern, high-performance web framework built specifically for APIs.[102] It is asynchronous (async/await) from the ground up and one of the fastest Python frameworks available, with performance on par with NodeJS and Go.[104] This is critical for a production-ready control-plane API.
- **Developer Velocity & Robustness:** FastAPI uses standard Python *type hints* and Pydantic models for automatic data validation, serialization, and conversion.[104] This dramatically reduces human-induced errors (by ~40%) and boilerplate code.[104]
- **Automatic Documentation:** FastAPI automatically generates interactive OpenAPI (Swagger) and ReDoc API documentation from the Pydantic models.[104] This directly fulfills the "API documentation" deliverable with no additional effort.
- **Ecosystem:** The framework is designed for production and integrates seamlessly with Docker, Kubernetes, and databases like PostgreSQL.[50]

3.3 API Schema for Zero-Trust Policies

The core of the IBN API is the JSON schema that defines the "intent".[111] This schema will be implemented as a Pydantic model in FastAPI [104] and must adhere to ZTA design principles, such as least privilege and microsegmentation.[115]

This schema is the "language" of our IBN, providing the API contract for all clients, including human operators and the ML service (defined in Part 4).

Table 2: Intent-Based API Policy JSON Schema Definition

Field Name	Data Type	Description	Example
id	string (uuid)	Unique policy identifier (read-only, server-generated).	"a1b2c3d4-..."

name	string	**Required.** Human-readable name for the policy.	"Isolate-Prod-DB-from-Dev"
priority	integer	Policy matching precedence (higher number = higher priority). Default: 1000.	5000
source	object	**Required.** The entity initiating the traffic. Must contain one of:	
.label_selector	object	K8s-style label selector.[92]	{"env": "prod", "app": "frontend"}
.ip_block	string (cidr)	IP CIDR block for non-K8s or external entities.[91]	"172.16.0.0/24"
destination	object	**Required.** The entity receiving the traffic. Must contain one of:	
.label_selector	object	K8s-style label selector.	{"env": "prod", "app": "db"}
.ip_block	string (cidr)	IP CIDR block.	"10.0.1.5/32"
service	array[object]	**Optional.** L4 rules. If omitted, applies to all protocols/ports.	
.protocol	string	**Required if service is present.** Enum: "TCP",	"TCP"

		"UDP", "ICMP".	
.port	integer	**Optional.** Target port (1-65535).[91]	5432
action	string	**Required.** The ZTA enforcement action. Enum: "ALLOW", "DENY".	"DENY"
status	string	**Required.** Toggles the policy. Enum: "ENABLED", "DISABLED".	"ENABLED"

This schema is declarative, hybrid (supporting both K8s-native label_selector and traditional ip_block), and maps directly to ZTA principles [61] and OpenFlow's matching capabilities.[80]

Part 4: Real-Time Analytics and ML-Driven Security

This section designs the analytics pipeline, incorporating real-time telemetry, a machine learning model for anomaly detection, and a visualization dashboard.

4.1 The Telemetry Collection Pipeline

The project requires "real-time monitoring" and "telemetry collection" [119] to feed both the visualization dashboard and the ML models. The choice of protocol is critical.

- **Protocol Analysis:** The main candidates are NetFlow/IPFIX, sFlow, and gNMI.[122]
  - NetFlow/IPFIX [122]: Statefully track *flows* (sessions) and export aggregate "flow records." While thorough, this can introduce delays as the flow cache must be filled or expire.[128]
  - sFlow [122]: Exports *statistically sampled packet headers* in near real-time.[127] It is stateless on the switch and provides full L2-L7 visibility into the samples.[124] This

real-time, header-level detail makes it ideal for *security analysis and anomaly detection*.[128]
    - gNMI [125]: A modern, model-driven (OpenConfig) streaming protocol using gRPC.[131] It allows a collector to *subscribe* to high-frequency counters (e.g., interface stats, CPU, queue depth).[100] It is ideal for *real-time performance monitoring*.
- **Architectural Decision:** A *hybrid* approach will be used.
    1. **sFlow:** Will be the primary data source for the ML security model, as it provides the packet-level detail needed for feature extraction.
    2. **gNMI:** Will be the primary data source for the Grafana performance dashboard, providing high-resolution, real-time metrics for network health.

Table 3: Telemetry Protocol Comparison and Selection

Protocol	Mechanism	Data Granularity	Primary Use Case in this Project
NetFlow/IPFIX	Stateful flow-caching [124]	L3/L4 Flow Records [124]	*Not used (sFlow is superior for security).*
sFlow	Stateless packet sampling [124]	L2-L7 Packet Headers [124]	**Machine Learning Feature Extraction** [130]
gNMI	Model-driven streaming (gRPC) [131]	High-frequency counters [100]	**Real-time Performance Dashboard** [132]

-

Implementation: A standalone, decoupled Python telemetry collector service will be built.
    1. **OVS Configuration:** All OVS bridges will be configured using ovs-vsctl to export sFlow datagrams to the collector's UDP port (e.g., 6343).[128]
    2. **gNMI Subscription:** The collector will use the pygnmi library [139] to create a gRPC client that *subscribes* to desired OpenConfig paths (e.g., /interfaces/interface/state/counters) on the network devices.[142]
    3. **sFlow Ingestion:** The collector will use a library like pysflow [145] or sflow-collector [146] to listen for and parse the incoming sFlow datagrams.
    4. **Metrics Export:** This service will parse both streams, aggregate the data, and expose it on a /metrics endpoint for Prometheus to scrape.[132]

4.2 Machine Learning for Anomaly Detection (DDoS)

This component fulfills the "DDoS mitigation" and "ML components" requirements by creating a system that detects anomalous traffic and automatically triggers a mitigation response.[149] The faucetsdn/poseidon project [152] and its NetworkML component [155] serve as an excellent real-world blueprint. This project demonstrates using an SDN controller (Faucet, which is Ryu-based [154]) to mirror traffic [153] to an ML model (NetworkML) that classifies device behavior.[155]

- **Model Selection:** The system requires a model suitable for *real-time, unsupervised* anomaly detection. While deep learning is an option [8], a more lightweight and faster model is preferable for real-time response.
- **Decision: Isolation Forest** is the selected model.[162] It is an unsupervised ML algorithm that is extremely fast, memory-efficient [166], and well-suited for high-dimensional data like network flows.[167] It works by "isolating" anomalies, which are statistically "easier" to separate from normal data points.[166] The implementation will use sklearn.ensemble.IsolationForest.[164]
- **Implementation (The Closed-Loop Feedback System):**
  1. **Feature Extraction:** The Telemetry Collector service (from 4.1) will parse sFlow data [130] and extract features for each flow (e.g., src_ip, dst_ip, protocol, packet_count, byte_count).[171] This feature set is streamed to a dedicated ml-analytics service.
  2. **Training:** The Isolation Forest model (iso_forest.fit()) [170] is trained on a "golden" dataset of normal network traffic to establish a baseline.
  3. **Real-Time Scoring:** The ml-analytics service takes live flow data, extracts features, and uses iso_forest.predict() [170] to generate an *anomaly score*. Scores of -1 indicate an anomaly.[166]

This design incorporates a highly flexible, *decoupled* architecture. The ml-analytics service does not need to know *anything* about OpenFlow or Ryu. It only needs to be a client of our own **Intent API (Part 3)**.

This "dogfooding" approach enables a fully automated, closed-loop mitigation:

1. The ml-analytics service (running Isolation Forest) detects a severe anomaly, such as a DDoS attack [173], from src_ip=1.2.3.4.
2. Instead of attempting to communicate with Ryu directly, the service acts as an authenticated API client.
3. It makes a standard POST request to our fastapi-api service (from 3.2):
   POST /api/v1/policies

Authorization: Bearer <ml_service_token>

```JSON
{
 "name": "ML-DDoS-Mitigation-1.2.3.4",
 "priority": 65000,
 "source": {"ip_block": "1.2.3.4/32"},
 "destination": {"ip_block": "0.0.0.0/0"},
 "action": "DENY",
 "status": "ENABLED"
}
```

4. The fastapi-api service validates this intent [104] and writes it to the PostgreSQL database (as per 1.3).
5. The **Master Ryu Controller** (which is not part of this transaction) detects the new, high-priority policy in the database (as per 2.3), translates it into an OpenFlow rule (priority=65000, ip_src=1.2.3.4, actions=DROP) [81], and installs it on *all* switches.

The DDoS attack is now blocked at the network edge within seconds. This is a fully automated, closed-loop [100], and elegantly decoupled microservice architecture, where the ML component is just another client of the IBN API.

4.3 Visualization and Monitoring

To fulfill the "Real-time visualization dashboard" requirement, a **Prometheus + Grafana** stack will be used.[132]

- **Data Pipeline:**
  1. **Prometheus:** Deployed as part of the stack, Prometheus will be configured to "scrape" metrics from:
     - The telemetry-collector service (gNMI and sFlow aggregates).[132]
     - The ryu-controller instances (OpenFlow stats, HA status, flow-mods/sec).[157]
     - The ml-analytics service (anomaly scores, mitigation events).
  2. **Grafana:** Deployed and pre-configured with Prometheus as its data source.[176]
- **Dashboards:** A set of custom Grafana dashboards will be created:
  - SDN Controller Dashboard [175]: Visualizes controller health (Master/Slave status), Packet-In rates, Flow-Mod rates, and connected switches.
  - Network Telemetry Dashboard [132]: Shows per-switch and per-port throughput, packet counts, and error rates (from gNMI). Also displays "Top-N Talkers" (from sFlow).

- ZTA Security Dashboard [173]: A real-time chart of the network anomaly score, a counter for blocked packets (from the high-priority DROP rules), and a table of all active automated mitigation policies.

Part 5: Deployment and Validation (The Deliverables)

This final section outlines the practical deliverables: containerization for easy deployment and a comprehensive benchmarking plan for validation.

5.1 Production-Ready Containerization (Docker)

To fulfill the "Docker containerization" deliverable, docker-compose will be used to define and manage the entire multi-container application stack.[108] This provides a "one-command" (docker-compose up) launch for development, testing, and production.

The docker-compose.yml file will define the following microservices [179]:

- zookeeper: A standard Zookeeper image for HA coordination.[181]
- postgres-db: A standard PostgreSQL image for policy persistence.[50]
- fastapi-api: A custom Dockerfile based on python:3.11-slim, installing FastAPI, Pydantic, and Uvicorn.[109]
- ryu-controller: A custom Dockerfile based on osrg/ryu [37] or a standard Python image [182], installing Ryu, kazoo [28], and the kubernetes client.[83]
- telemetry-collector: A custom Dockerfile installing pysflow [145] and pygnmi.[139]
- ml-analytics: A custom Dockerfile installing scikit-learn [170] and requests.
- prometheus: The standard prom/prometheus image, with a mounted prometheus.yml configuration file.[184]
- grafana: The standard grafana/grafana image, with provisioned data sources.[184]

This docker-compose.yml file *is* the architectural specification, defining the microservice boundaries and dependencies, which is critical for a "production-ready" system.

5.2 Benchmarking Methodology (Mininet)

To fulfill the "performance benchmarking results" deliverable, **Mininet** will be used. Mininet is the de-facto standard for emulating SDN topologies and controllers in a realistic, high-fidelity environment.[12]

A custom Mininet Python script (e.g., kube_topo.py) will be written to emulate the target environment [191]:

- It will create N OVSKernelSwitch instances [195] to represent the OVS bridges on K8s Worker Nodes.
- It will create Mininet host instances to simulate Pods, connecting them to their respective node-switches.[194]
- It will set the controller for *all* switches to point to the *remote* ryu-controller service cluster running in Docker Compose.[191]

5.3 Performance and Security Benchmarking (Test Cases)

A matrix of specific tests will be executed to validate all functional and non-functional requirements.[12] The most critical benchmark is not just raw performance, but the *performance overhead* of the security model.

The test plan will measure the *delta* in performance between the baseline network and the secured network.

- **Test A (Baseline):** Run the Mininet topology with a simple L2 switching app [191] (emulating the "default-allow" CNI). Run iperf h1 h2 [12] and ping tests to record baseline throughput and latency.
- **Test B (ZTA Overhead):** Run the *same* topology, but pointed to the full Ryu HA cluster. POST a single "ALLOW ALL" policy. Re-run iperf and ping tests. The delta (Time(B) - Time(A)) measures the *base overhead* of the ZTA controller.
- **Test C (ZTA Scaling):** POST 1,000 "DENY" rules. Re-run iperf and ping. The delta (Time(C) - Time(B)) measures the *policy scaling cost* of the solution. This is an expert-level benchmark.[12]

Table 4: Benchmarking Test Matrix and Expected Results

Test ID	Test Case	Tools Used	Metric	Expected

				Outcome
1.	**Baseline Performance**	Mininet, iperf, ping [12]	Throughput (Gbps), Latency (ms)	Establishes the performance baseline (e.g., 9.4 Gbps, 0.2 ms).
2.	**ZTA Controller Overhead**	Mininet, iperf, ping	% change in Throughput/Latency vs. Baseline	Quantifies the base performance cost of the reactive controller (e.g., <5% impact).
3.	**Policy Enforcement (ZTA)**	Mininet, ping, curl	Pass / Fail	ping h1 h2 succeeds. POST /policy (DENY) via curl. ping h1 h2 now fails.
4.	**ML-Driven DDoS Mitigation**	Mininet, hping3, curl, Grafana [198]	Time-to-Mitigation (sec)	Start hping3 flood. Grafana dashboard shows anomaly. curl to a victim host fails. System automatically installs DROP rule. curl resumes.
5.	**HA Controller Failover**	Mininet, iperf -t 300, docker kill [21]	Packet Loss (%)	iperf stream is active. docker kill the Master Ryu container. iperf stream

				freezes, Zookeeper elects new Master [35], and stream resumes within <1 second.

5.4 API and Code Documentation Strategy

The "comprehensive documentation" and "API documentation" deliverables will be met using a combination of automated and manual methods.

- **API Documentation (Automated):** This is a primary benefit of selecting FastAPI.[104] By defining the API schema (from 3.3) using Pydantic models, FastAPI automatically generates and hosts:
  - /docs: An interactive Swagger UI for API exploration and testing.[106]
  - /redoc: Alternative ReDoc documentation for a clean, static view.[106]
- **Code Documentation (Manual):**
  - **Ryu Application:** Comprehensive Python docstrings, type hinting, and a README.md file explaining the event-driven logic (e.g., the EventK8sPodUpdate bridge) and the OpenFlow priority override model.[78]
  - **Deployment Documentation:** The docker-compose.yml [179] and Mininet benchmarking scripts [194] will be heavily commented, serving as the primary guides for setup, deployment, and validation.

Part 6: Concluding Architectural Summary

This report provides the complete architectural blueprint for a production-ready, Zero-Trust security framework. The solution integrates a stack of modern, open-source, and Python-native technologies to solve a complex, real-world challenge at the intersection of SDN, cloud-native environments, and automated security.

The core architectural tenets are:

1. **Modular Control:** Using **Ryu** as a flexible, Python-native "toolbox" controller.[4]
2. **Transparent HA:** Implementing a robust Master/Slave HA model using **Zookeeper**

(kazoo) for leader election [4] and OpenFlow RoleRequest for data-plane failover.[38]

3. **Decoupled Intent:** Using a **PostgreSQL** database as the "source of truth" for policy [48], decoupling the **FastAPI**-based IBN API [100] from the Ryu controller.
4. **Non-Intrusive Integration:** Co-existing with the K8s OVN-Kubernetes CNI [70] by using a **Priority-Override** OpenFlow model [78] to layer ZTA rules *over* default CNI connectivity.
5. **Event-Driven Reconciliation:** Using the **kubernetes-client** watch [84] to translate K8s label [92] changes into flow rules in real-time, bridging K8s events to custom Ryu events.
6. **Hybrid Telemetry:** Implementing a hybrid telemetry pipeline with **sFlow** [136] for security and **gNMI** [139] for performance, feeding an **Isolation Forest** ML model [166] for anomaly detection.
7. **Automated Mitigation:** Enabling the ML model to trigger a DROP policy by *calling its own IBN API* [173], creating a fully automated, closed-loop mitigation system.
8. **Full Observability:** Visualizing the entire stack's health, performance, and security posture using **Prometheus and Grafana**.[132]
9. **Turnkey Deployment:** Packaging the entire microservice-based application using **Docker Compose** [179] and validating all requirements using **Mininet**.[191]

Works cited

1. SDN Controller Comparison 2025: ONOS vs ODL vs Ryu Guide - Aptira, accessed November 7, 2025, https://aptira.com/sdn-controller-comparison/
2. An Overview of SDN Issues—A Case Study and Performance Evaluation of a Secure OpenFlow Protocol Implementation - MDPI, accessed November 7, 2025, https://www.mdpi.com/2079-9292/14/16/3244
3. Controller selection in software defined networks using best-worst multi-criteria decision-making, accessed November 7, 2025, https://beei.org/index.php/EEI/article/download/2393/2020
4. Ryu SDN Controller Review: Python-Based Flexible Framework - Aptira, accessed November 7, 2025, https://aptira.com/ryu-sdn-controller-review/
5. ryu Documentation - Read the Docs, accessed November 7, 2025, https://buildmedia.readthedocs.org/media/pdf/ryu/latest/ryu.pdf
6. ryu Documentation, accessed November 7, 2025, https://ryu.readthedocs.io/_/downloads/en/latest/pdf/
7. hainamt/Anomaly_dectection_ML_SDN_Ryu: A prototype for an architecture that uses Machine Learning to detect abnormalities in the SDN network packet flow, built by Ryu SDN and mininet - GitHub, accessed November 7, 2025, https://github.com/hainamt/Anomaly_dectection_ML_SDN_Ryu
8. (PDF) Application-Based Online Traffic Classification with Deep Learning Models on SDN Networks - ResearchGate, accessed November 7, 2025, https://www.researchgate.net/publication/343155555_Application-Based_Online_Traffic_Classification_with_Deep_Learning_Models_on_SDN_Networks
9. Performance Comparison Of SDN Controllers In Different Network Topologies - Research Square, accessed November 7, 2025,

https://assets-eu.researchsquare.com/files/rs-4826985/v2_covered_fc94ca8b-ee71-4fde-bacf-f486a761bf97.pdf
10. (PDF) Performance Comparison Of SDN Controllers In Different Network Topologies, accessed November 7, 2025, https://www.researchgate.net/publication/382909004_Performance_Comparison_Of_SDN_Controllers_In_Different_Network_Topologies/download
11. A Qualitative and Comparative Performance Assessment of Logically Centralized SDN Controllers by Mininet Emulator - Preprints.org, accessed November 7, 2025, https://www.preprints.org/manuscript/202401.2116/v1/download
12. A Qualitative and Comparative Performance Assessment of Logically Centralized SDN Controllers via Mininet Emulator - MDPI, accessed November 7, 2025, https://www.mdpi.com/2073-431X/13/4/85
13. Performance Analysis of Various SDN Controllers with Different Network Size in SDWN - EUDL, accessed November 7, 2025, https://eudl.eu/pdf/10.4108/eai.7-9-2021.2314875
14. Open Source Software-Defined Networking Controllers—Operational and Security Issues, accessed November 7, 2025, https://www.mdpi.com/2079-9292/13/12/2329
15. onosproject/onos - Docker Image, accessed November 7, 2025, https://hub.docker.com/r/onosproject/onos/
16. Design and deployment of secure, robust, and resilient SDN Controllers - Queen's University Belfast, accessed November 7, 2025, https://pure.qub.ac.uk/files/17774519/Design_and_Deployment.pdf
17. Ryu Network Operating System, accessed November 7, 2025, https://events.static.linuxfound.org/images/stories/pdf/lcna_co2012_ohmura.pdf
18. A Fault-Tolerant and Consistent SDN Controller - AGH, accessed November 7, 2025, https://galaxy.agh.edu.pl/~andrzejk/pub/2016_globecom_sdn_controller.pdf
19. Database Engineering Part 7: Replication and High Availability Strategies in Database Systems | by Augustine Umeagudosi | Medium, accessed November 7, 2025, https://medium.com/@augustineumeagudosi/database-engineering-part-7-replication-and-high-availability-strategies-in-database-systems-292d1dd7d610
20. ZooKeeper Recipes and Solutions, accessed November 7, 2025, https://zookeeper.apache.org/doc/current/recipes.html
21. Ravana: Controller Fault-Tolerance in Software-Defined Networking - Duke Computer Science, accessed November 7, 2025, https://courses.cs.duke.edu/spring17/compsci590.7/Papers/Ravana15.pdf
22. Byzantine Fault Tolerant Software-Defined Networking (SDN) Controllers - SPROUT, accessed November 7, 2025, https://sprout.ics.uci.edu/pubs/resilient_sdn_controller.pdf
23. A new approach in fault tolerance in control level of SDN, accessed November 7, 2025, https://ijnaa.semnan.ac.ir/article_9028_a3b555ea1eb75a92e2b3dbd46a22f2d5.pdf
24. Distributed Coordination services (ZooKeeper) - System design -

GeeksforGeeks, accessed November 7, 2025, https://www.geeksforgeeks.org/system-design/distributed-coordination-services-zookeeper-system-design/
25. Ryu Controller | PDF | Component Based Software Engineering | Computer Architecture, accessed November 7, 2025, https://www.scribd.com/document/250915607/Ryu-controller
26. Exploring Performance of etcd, Zookeeper and Consul Consistent Key-value Datastores : r/programming - Reddit, accessed November 7, 2025, https://www.reddit.com/r/programming/comments/5ux3ya/exploring_performance_of_etcd_zookeeper_and/
27. Leader election in a Distributed System Using ZooKeeper - GeeksforGeeks, accessed November 7, 2025, https://www.geeksforgeeks.org/system-design/leader-election-in-a-distributed-system-using-zookeeper/
28. kazoo.recipe.election — kazoo 2.10.0 documentation - Read the Docs, accessed November 7, 2025, https://kazoo.readthedocs.io/en/latest/api/recipe/election.html
29. How to use kazoo client for leader election? - Stack Overflow, accessed November 7, 2025, https://stackoverflow.com/questions/39125064/how-to-use-kazoo-client-for-leader-election
30. kazoo-sasl 2.6.1 - PyPI, accessed November 7, 2025, https://pypi.org/project/kazoo-sasl/
31. Navigating the Jungle of Distributed Systems: A Guide to ZooKeeper and Leader Election Algorithms - Hewi's Blog, accessed November 7, 2025, https://hewi.blog/navigating-the-jungle-of-distributed-systems-a-guide-to-zookeeper-and-leader-election-algorithms
32. ZooKeeper Recipes and Solutions, accessed November 7, 2025, https://zookeeper.apache.org/doc/r3.1.2/recipes.html
33. kazoo Documentation, accessed November 7, 2025, https://kazoo.readthedocs.io/_/downloads/en/2.4.0/pdf/
34. kazoo.client — kazoo 2.10.0 documentation - Read the Docs, accessed November 7, 2025, https://kazoo.readthedocs.io/en/latest/api/client.html
35. Leader Election and Failover with Zookeeper - Case Study - Vasil Kosturski, accessed November 7, 2025, https://vkontech.com/leader-election-and-failover-with-zookeeper-case-study/
36. Leader election in a distributed system using ZooKeeper | by Minhazul Hayat Khan, accessed November 7, 2025, https://medium.com/@minhaz1217/leader-election-in-a-distributed-system-using-zookeeper-b562e6d79855
37. SDN Framework - GitHub Pages, accessed November 7, 2025, https://osrg.github.io/ryu-book/en/Ryubook.pdf
38. OpenFlow v1.4 Messages and Structures - Ryu - Read the Docs, accessed November 7, 2025, https://ryu.readthedocs.io/en/latest/ofproto_v1_4_ref.html
39. OpenFlow v1.3 Messages and Structures — Ryu 4.34 documentation, accessed November 7, 2025, https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html

40. Because Coordinating Distributed Systems is a Zoo - ZooKeeper, accessed November 7, 2025, https://zookeeper.apache.org/doc/r3.7.2/zookeeperOver.html
41. Watches and Ephemeral node doesn't work when state of zookeeper changes automatically? - Stack Overflow, accessed November 7, 2025, https://stackoverflow.com/questions/20171418/watches-and-ephemeral-node-doesnt-work-when-state-of-zookeeper-changes-automati
42. Basic Usage — kazoo 2.10.0 documentation - Read the Docs, accessed November 7, 2025, https://kazoo.readthedocs.io/en/latest/basic_usage.html
43. kazoo.recipe.watchers — kazoo 2.10.0 documentation - Read the Docs, accessed November 7, 2025, https://kazoo.readthedocs.io/en/latest/api/recipe/watchers.html
44. How to watch for events on the descendant nodes in ZooKeeper using kazoo? · Issue #145, accessed November 7, 2025, https://github.com/python-zk/kazoo/issues/145
45. connect openvswitch to several controller - Google Groups, accessed November 7, 2025, https://groups.google.com/a/onosproject.org/g/onos-dev/c/HfFglH9gvt0
46. SDN Lab with Ryu controller, accessed November 7, 2025, https://suhu0426.github.io/Web/Presentation/20141021/index.html
47. OpenFlow Switch Specification - Open Networking Foundation, accessed November 7, 2025, https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf
48. SDN architecture - Open Networking Foundation, accessed November 7, 2025, https://opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf
49. If you could choose any Python web framework to build APIs for a startup, which one would you choose and why? - Reddit, accessed November 7, 2025, https://www.reddit.com/r/Python/comments/xs7s6a/if_you_could_choose_any_python_web_framework_to/
50. Full stack, modern web application template. Using FastAPI, React, SQLModel, PostgreSQL, Docker, GitHub Actions, automatic HTTPS and more., accessed November 7, 2025, https://github.com/fastapi/full-stack-fastapi-template
51. Ravana: Controller Fault-Tolerance in Software-Defined Networking - Naga Katta, accessed November 7, 2025, https://nkatta.github.io/papers/ravana15.pdf
52. What Is Intent-Based Networking (IBN)? Definition & How It Works - Nile Secure, accessed November 7, 2025, https://nilesecure.com/ai-networking/what-is-intent-based-networking-ibn-definition-how-it-works
53. What is the NIST SP 800-207 cybersecurity framework? - CyberArk, accessed November 7, 2025, https://www.cyberark.com/what-is/nist-sp-800-207-cybersecurity-framework/
54. Understanding NIST 800-207 - Blog - RiskRecon, accessed November 7, 2025, https://blog.riskrecon.com/understanding-nist-800-207
55. What is Zero Trust? - Guide to Zero Trust Security - CrowdStrike, accessed November 7, 2025,

https://www.crowdstrike.com/en-us/cybersecurity-101/zero-trust-security/
56. Zero Trust Architecture - NIST Technical Series Publications, accessed November 7, 2025, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf
57. SP 800-207A, A Zero Trust Architecture Model for Access Control in Cloud-Native Applications in Multi-Cloud Environments - NIST Computer Security Resource Center, accessed November 7, 2025, https://csrc.nist.gov/pubs/sp/800/207/a/final
58. Microsegmentation and Zero Trust: A Powerful Security Duo - Elisity, accessed November 7, 2025, https://www.elisity.com/blog/microsegmentation-and-zero-trust
59. Zero Trust Network Segmentation: Guide & Best Practices - Nile Secure, accessed November 7, 2025, https://nilesecure.com/network-design/zero-trust-network-segmentation
60. What Is Microsegmentation? - Palo Alto Networks, accessed November 7, 2025, https://www.paloaltonetworks.com/cyberpedia/what-is-microsegmentation
61. The Journey to Zero Trust: Microsegmentation in Zero Trust ... - CISA, accessed November 7, 2025, https://www.cisa.gov/sites/default/files/2025-07/ZT-Microsegmentation-Guidance-Part-One_508c.pdf
62. Advancing Zero Trust Maturity Throughout the Network and Environment Pillar - DoD, accessed November 7, 2025, https://media.defense.gov/2024/Mar/05/2003405462/-1/-1/0/CSI-ZERO-TRUST-NETWORK-ENVIRONMENT-PILLAR.PDF
63. Secure networks with SASE, Zero Trust, and AI - Microsoft Learn, accessed November 7, 2025, https://learn.microsoft.com/en-us/security/zero-trust/deploy/networks
64. Installing Addons | Kubernetes, accessed November 7, 2025, https://kubernetes.io/docs/concepts/cluster-administration/addons/
65. Chapter 24. OpenShift SDN network plugin - Red Hat Documentation, accessed November 7, 2025, https://docs.redhat.com/en/documentation/openshift_container_platform/4.12/html/networking/openshift-sdn-network-plugin
66. OVN-Kubernetes Architecture, accessed November 7, 2025, https://ovn-kubernetes.io/design/architecture/
67. OVN-Kubernetes architecture - Networking - OKD Documentation, accessed November 7, 2025, https://docs.okd.io/4.13/networking/ovn_kubernetes_network_provider/ovn-kubernetes-architecture-assembly.html
68. How the new OVN Kubernetes architecture helps with scaling OpenShift - Red Hat, accessed November 7, 2025, https://www.redhat.com/en/blog/how-the-new-ovn-kubernetes-architecture-helps-with-scaling-openshift
69. Chapter 1. About the OVN-Kubernetes network plugin - Red Hat Documentation, accessed November 7, 2025, https://docs.redhat.com/en/documentation/red_hat_build_of_microshift/4.14/html/

networking/microshift-cni
70. Chapter 19. OVN-Kubernetes network plugin | Networking | OpenShift Container Platform | 4.16 | Red Hat Documentation, accessed November 7, 2025, https://docs.redhat.com/en/documentation/openshift_container_platform/4.16/html/networking/ovn-kubernetes-network-plugin
71. Chapter 25. OVN-Kubernetes network plugin | Networking | OpenShift Container Platform | 4.13 | Red Hat Documentation, accessed November 7, 2025, https://docs.redhat.com/en/documentation/openshift_container_platform/4.13/html/networking/ovn-kubernetes-network-plugin
72. Chapter 16. OVN-Kubernetes default CNI network provider | Networking | OpenShift Container Platform | 4.8 | Red Hat Documentation, accessed November 7, 2025, https://docs.redhat.com/en/documentation/openshift_container_platform/4.8/html/networking/ovn-kubernetes-default-cni-network-provider
73. A brief overview of the Container Network Interface (CNI) in Kubernetes - Red Hat, accessed November 7, 2025, https://www.redhat.com/it/blog/cni-kubernetes
74. Introduction to OVN-Kubernetes CNI in Kubernetes | by Mohitverma - Medium, accessed November 7, 2025, https://medium.com/@mohitverma160288/introduction-to-ovn-kubernetes-cni-in-kubernetes-aff766d6a2f0
75. Chapter 2. OVN-Kubernetes architecture - Red Hat Documentation, accessed November 7, 2025, https://docs.redhat.com/en/documentation/openshift_container_platform/4.19/html/ovn-kubernetes_network_plugin/ovn-kubernetes-architecture-assembly
76. Getting Started — Ryu 4.34 documentation - Read the Docs, accessed November 7, 2025, https://ryu.readthedocs.io/en/latest/parameters.html
77. Network OS Software Defined Networking (SDN) Configuration Guide for release 6.0.1a - Product Documentation, accessed November 7, 2025, https://documentation.extremenetworks.com/networkos/SW/60x/53-1003782-02_SDNNetworkOS_6.0.1a_CG_Sep2015.pdf
78. Software Defined Networking (SDN) with OpenFlow 1.3, Open vSwitch and Ryu, accessed November 7, 2025, https://hepia.infolibre.ch/Virtualisation-Reseaux/sdn_with_openflow1.3_ovswitch_and_ryu.pdf
79. OpenFlow v1.5 Messages and Structures — Ryu 4.34 documentation - Read the Docs, accessed November 7, 2025, https://ryu.readthedocs.io/en/latest/ofproto_v1_5_ref.html
80. Switching Hub — Ryubook 1.0 documentation - GitHub Pages, accessed November 7, 2025, https://osrg.github.io/ryu-book/en/html/switching_hub.html
81. Ryu Controller Drop Packet - Stack Overflow, accessed November 7, 2025, https://stackoverflow.com/questions/41023354/ryu-controller-drop-packet
82. How to Forward all traffic to controller in Ryu/OpenFlow? : r/networking - Reddit, accessed November 7, 2025, https://www.reddit.com/r/networking/comments/4hj6tm/how_to_forward_all_traffic_to_controller_in/

83. A Beginner's Guide to Kubernetes Python Client - Velotio Technologies, accessed November 7, 2025, https://www.velotio.com/engineering-blog/kubernetes-python-client
84. Access Clusters Using the Kubernetes API, accessed November 7, 2025, https://kubernetes.io/docs/tasks/administer-cluster/access-cluster-api/
85. Practical Guide to Kubernetes with Python - Plural, accessed November 7, 2025, https://www.plural.sh/blog/python-kubernetes-guide/
86. Kubernetes Python Client Running in Cluster - A-Team Chronicles, accessed November 7, 2025, https://www.ateam-oracle.com/post/kubernetes-python-client-running-in-cluster
87. Examples show off `load_kube_config()` as the Right Way to set up the module, but an attempt at `load_incluster_config()` is also required to match the behavior of kubectl and work form inside pods · Issue #1005 · kubernetes-client/python - GitHub, accessed November 7, 2025, https://github.com/kubernetes-client/python/issues/1005
88. kubernetes.config package, accessed November 7, 2025, https://jashandeep-sohik8s-python.readthedocs.io/en/latest/kubernetes.config.html
89. accessing kubernetes python api through a pod - Stack Overflow, accessed November 7, 2025, https://stackoverflow.com/questions/51092807/accessing-kubernetes-python-api-through-a-pod
90. Authenticating | Kubernetes, accessed November 7, 2025, https://kubernetes.io/docs/reference/access-authn-authz/authentication/
91. Network Policies - Kubernetes, accessed November 7, 2025, https://kubernetes.io/docs/concepts/services-networking/network-policies/
92. Labels and Selectors - Kubernetes, accessed November 7, 2025, https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
93. Kubernetes Labels and Selectors: A Definitive Guide with Hands-on - Devtron, accessed November 7, 2025, https://devtron.ai/blog/kubernetes-labels-and-selectors-a-definitive-guide-with-hands-on/
94. `list_namespaced_pod()` should accept `label_selector` values of `dict` type · Issue #2133 · kubernetes-client/python - GitHub, accessed November 7, 2025, https://github.com/kubernetes-client/python/issues/2133
95. REST API – Programming the SDN - IP Files - WordPress.com, accessed November 7, 2025, https://ipfiles.wordpress.com/2014/01/05/rest-api-programming-the-sdn/
96. "Integrating APIs with SDN Networks: Streamlining Network Management" - YouTube, accessed November 7, 2025, https://www.youtube.com/watch?v=Jdw1dVRh5ds
97. What is Intent-Based Networking? | Glossary | HPE, accessed November 7, 2025, https://www.hpe.com/us/en/what-is/intent-based-networking.html
98. (PDF) INTENT-BASED NETWORKING IN SDN: AUTOMATING NETWORK

CONFIGURATION AND MANAGEMENT - ResearchGate, accessed November 7, 2025, https://www.researchgate.net/publication/393185156_INTENT-BASED_NETWORKING_IN_SDN_AUTOMATING_NETWORK_CONFIGURATION_AND_MANAGEMENT
99. Intent-Based Networking (IBN) — Moving from Automation to Autonomy - Medium, accessed November 7, 2025, https://medium.com/@RocketMeUpNetworking/intent-based-networking-ibn-moving-from-automation-to-autonomy-288f7cdf1270
100. Intent-Based Network Automation in Modern Infrastructure - ACE Journal, accessed November 7, 2025, https://www.acejournal.org/2025/06/05/intent-based-network-automation
101. Intent-based RESTful APIs | Ribbon Communications, accessed November 7, 2025, https://ribboncommunications.com/company/media-center/blog/intent-based-restful-apis
102. Top 10 Python REST API Frameworks in 2024 | BrowserStack, accessed November 7, 2025, https://www.browserstack.com/guide/top-python-rest-api-frameworks
103. FastAPI vs. Flask: Python web frameworks comparison and tutorial - Contentful, accessed November 7, 2025, https://www.contentful.com/blog/fastapi-vs-flask/
104. FastAPI, accessed November 7, 2025, https://fastapi.tiangolo.com/
105. A Close Look at a FastAPI Example Application - Real Python, accessed November 7, 2025, https://realpython.com/fastapi-python-web-apis/
106. FastAPI framework, high performance, easy to learn, fast to code, ready for production - GitHub, accessed November 7, 2025, https://github.com/fastapi/fastapi
107. Deployments Concepts - FastAPI, accessed November 7, 2025, https://fastapi.tiangolo.com/deployment/concepts/
108. FastAPI in Containers - Docker, accessed November 7, 2025, https://fastapi.tiangolo.com/deployment/docker/
109. Deploying a FastAPI Application on Kubernetes: A Step-by-Step Guide for Production, accessed November 7, 2025, https://sumanta9090.medium.com/deploying-a-fastapi-application-on-kubernetes-a-step-by-step-guide-for-production-d74faac4ca36
110. Developing FastAPI Application using K8s & AWS - JetBrains Guide, accessed November 7, 2025, https://www.jetbrains.com/guide/python/tutorials/fastapi-aws-kubernetes/
111. 3 Working with the Programmatic Intent-Based Network REST API, accessed November 7, 2025, https://docs.oracle.com/cd/E75627_01/doc.734/e75634/api_rest.htm
112. OpenAPI Specification - Version 3.1.0 - Swagger, accessed November 7, 2025, https://swagger.io/specification/
113. Is there any standard for JSON API response format? - Stack Overflow, accessed November 7, 2025,

https://stackoverflow.com/questions/12806386/is-there-any-standard-for-json-api-response-format
114. Policy Format :: Documentation - con terra products, accessed November 7, 2025, https://docs.conterra.de/en/securitymanager-next/1.3/reference/policy-format.html
115. REST API Security: 4 Design Principles and 10 Essential Practices | CyCognito, accessed November 7, 2025, https://www.cycognito.com/learn/api-security/rest-api-security.php
116. What are Zero Trust APIs? - A10 Networks, accessed November 7, 2025, https://www.a10networks.com/blog/what-are-zero-trust-apis/
117. Designing ZTNA access policies for Cloudflare Access, accessed November 7, 2025, https://developers.cloudflare.com/reference-architecture/design-guides/designing-ztna-access-policies/
118. Building APIs with a Zero Trust Policy: Protecting Your Data Like Your Peace, accessed November 7, 2025, https://dev.to/devbalop/building-apis-with-a-zero-trust-policy-protecting-your-data-like-your-peace-2ifg
119. Network Telemetry Streaming Services in SDN-Based Disaggregated Optical Networks - SciSpace, accessed November 7, 2025, https://scispace.com/pdf/network-telemetry-streaming-services-in-sdn-based-4644ymqhye.pdf
120. Distributed intelligence for pervasive optical network telemetry - Optica Publishing Group, accessed November 7, 2025, https://opg.optica.org/abstract.cfm?uri=jocn-15-9-676
121. SDN-Enabled Resiliency, Monitoring and Control in Computer Networks - DiVA portal, accessed November 7, 2025, https://www.diva-portal.org/smash/get/diva2:1377732/FULLTEXT02
122. Network Telemetry Guide – Applications, How it's Deployed and Challenges, accessed November 7, 2025, https://www.parkplacetechnologies.com/blog/network-telemetry-guide-applications-deployment-challenges/
123. What is Network Telemetry? - Splunk, accessed November 7, 2025, https://www.splunk.com/en_us/blog/learn/network-telemetry.html
124. NetFlow vs. sFlow: What's the Difference? | Kentik Blog, accessed November 7, 2025, https://www.kentik.com/blog/netflow-vs-sflow/
125. Telemetry vs netflow vs sflow vs flow telemetry : r/networking - Reddit, accessed November 7, 2025, https://www.reddit.com/r/networking/comments/1ejtx9e/telemetry_vs_netflow_vs_sflow_vs_flow_telemetry/
126. RFC 9232 - Network Telemetry Framework - IETF Datatracker, accessed November 7, 2025, https://datatracker.ietf.org/doc/rfc9232/
127. Network Flow Monitoring Explained: NetFlow vs sFlow vs IPFIX - Varonis, accessed November 7, 2025, https://www.varonis.com/blog/flow-monitoring

128. Traffic visibility and control with sFlow - Open vSwitch, accessed November 7, 2025, http://www.openvswitch.org/support/ovscon2014/17/1400-ovs-sflow.pdf
129. What are Netflow and sFlow? | InfluxData, accessed November 7, 2025, https://www.influxdata.com/what-are-netflow-and-sflow/
130. A systematic literature review of unsupervised learning algorithms for anomalous traffic detection based on flows - arXiv, accessed November 7, 2025, https://arxiv.org/html/2503.08293v1
131. GRPC-BASED SDN CONTROL AND TELEMETRY FOR SOFT-FAILURE DETECTION OF SPECTRAL/SPACIAL SUPERCHANNELS - Zenodo, accessed November 7, 2025, https://zenodo.org/record/3631435/files/GRPC-BASED%20SDN%20CONTROL%20AND%20TELEMETRY.pdf
132. Top 5 Telemetry and Observability Tools for Supporting SONiC Networks, accessed November 7, 2025, https://be-net.com/sonic-network-observability-and-monitoring/
133. Get Started With gRPC | gNMI Tutorial | Telemetry | Protocol Buffer | Part 7 - YouTube, accessed November 7, 2025, https://www.youtube.com/watch?v=a7vdDITEWiA
134. gRPC Network Management Interface (gNMI) specification - OpenConfig, accessed November 7, 2025, https://www.openconfig.net/docs/gnmi/gnmi-specification/
135. A Guide to GNMI & gRPC: How Are They Revolutionizing Network Management?, accessed November 7, 2025, https://thinkpalm.com/blogs/a-guide-to-gnmi-grpc-how-are-they-revolutionizing-network-management/
136. Monitoring VM Traffic Using sFlow - Open vSwitch Documentation, accessed November 7, 2025, https://docs.openvswitch.org/en/latest/howto/sflow/
137. Open vSwitch - sFlow, accessed November 7, 2025, https://blog.sflow.com/2010/01/open-vswitch.html
138. Open vSwitch Network Monitoring Using sFlow and sFlow-RT - Adel N. Toosi's Blog, accessed November 7, 2025, https://adelnadjarantoosi.wordpress.com/2017/10/16/open-vswitch-network-monitoring-using-sflow-and-sflow-rt/
139. gnmi-py - PyPI, accessed November 7, 2025, https://pypi.org/project/gnmi-py/
140. akarneliuk/pygnmi: The pure Python implementation of the gNMI client. - GitHub, accessed November 7, 2025, https://github.com/akarneliuk/pygnmi
141. pygnmi - Open Management, accessed November 7, 2025, https://aristanetworks.github.io/openmgmt/examples/pygnmi/
142. Guidelines for Telemetry Data Subscriptions over gNMI | Junos OS | Juniper Networks, accessed November 7, 2025, https://www.juniper.net/documentation/us/en/software/junos/interfaces-telemetry/topics/concept/subscribe-rpc.html
143. From Python to Go 018. Interaction With Network Devices Using GNMI. - Karneliuk, accessed November 7, 2025, https://karneliuk.com/2025/04/from-python-to-go-018-interaction-with-network

-devices-using-gnmi/
144. 27. pygnmi to subscribe data and update the configuration - YouTube, accessed November 7, 2025, https://www.youtube.com/watch?v=PlUpcKXmxvU
145. eth2-networks/pysflow: sFlow parser, written in Python - GitHub, accessed November 7, 2025, https://github.com/eth2-networks/pysflow
146. auspex-labs/sflow-collector: A Python class for parsing sFlow packets. - GitHub, accessed November 7, 2025, https://github.com/auspex-labs/sflow-collector
147. pvanstam/python-sflow: Python library for decoding en coding sFlow data - GitHub, accessed November 7, 2025, https://github.com/pvanstam/python-sflow
148. Instructions to set up Prometheus and Grafana to monitor an SDN controlled by ONOS. - GitHub, accessed November 7, 2025, https://github.com/querciak/SDN-prometheus
149. Anomaly Detection in Software-Defined Networking Using Machine Learning - DOAJ, accessed November 7, 2025, https://doaj.org/article/64bbfc2fa7fe4ff9823eb2af61f8d9df?
150. A Machine Learning-Based Anomaly Prediction Service for Software-Defined Networks, accessed November 7, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9658740/
151. Effects of Machine Learning Approach in Flow-Based Anomaly Detection on Software-Defined Networking - MDPI, accessed November 7, 2025, https://www.mdpi.com/2073-8994/12/1/7
152. sdn · GitHub Topics, accessed November 7, 2025, https://github.com/topics/sdn
153. faucetsdn/poseidon: Poseidon is a python-based application that leverages software defined networks (SDN) to acquire and then feed network traffic to a number of machine learning techniques. The machine learning algorithms classify and predict the type of device. - GitHub, accessed November 7, 2025, https://github.com/faucetsdn/poseidon
154. Faucet SDN - GitHub, accessed November 7, 2025, https://github.com/faucetsdn
155. faucetsdn/NetworkML: Machine learning plugins for network traffic - GitHub, accessed November 7, 2025, https://github.com/faucetsdn/NetworkML
156. poseidon · GitHub Topics, accessed November 7, 2025, https://github.com/topics/poseidon
157. faucetsdn/faucet: FAUCET is an OpenFlow controller for multi table OpenFlow 1.3 switches, that implements layer 2 switching, VLANs, ACLs, and layer 3 IPv4 and IPv6 routing. - GitHub, accessed November 7, 2025, https://github.com/faucetsdn/faucet
158. Poseidon: A Machine Learning Approach to Network Device Role and Behavior Identification - SC18, accessed November 7, 2025, https://sc18.supercomputing.org/app/uploads/2018/11/SC18-NRE-027.pdf
159. How to deploy or integrate the h5 model and run it inside the ryu controller? - Reddit, accessed November 7, 2025, https://www.reddit.com/r/networking/comments/xtfp5l/how_to_deploy_or_integra

te_the_h5_model_and_run/
160. Application-Based Online Traffic Classification with Deep Learning Models on SDN Networks - Taiwan Association of Engineering and Technology Innovation, accessed November 7, 2025, https://ojs.imeti.org/index.php/AITI/article/view/4286/992
161. A lightweight anomaly detection model for network traffic using multi scale spatio temporal residual learning - NIH, accessed November 7, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC12215067/
162. Anomaly Detection for Network Traffic | Isolation Forest + Autoencoder (Python Flask Project) - YouTube, accessed November 7, 2025, https://www.youtube.com/watch?v=Ui5xP-B2nZ4
163. Efficient Real-Time Anomaly Detection in IoT Networks Using One-Class Autoencoder and Deep Neural Network - MDPI, accessed November 7, 2025, https://www.mdpi.com/2079-9292/14/1/104
164. Web Traffic Anomaly Detection Using Isolation Forest - MDPI, accessed November 7, 2025, https://www.mdpi.com/2227-9709/11/4/83
165. Comparing Autoencoder and Isolation Forest in Network Anomaly Detection - ResearchGate, accessed November 7, 2025, https://www.researchgate.net/publication/371407967_Comparing_Autoencoder_and_Isolation_Forest_in_Network_Anomaly_Detection
166. Anomaly Detection in Python with Isolation Forest - DigitalOcean, accessed November 7, 2025, https://www.digitalocean.com/community/tutorials/anomaly-detection-isolation-forest
167. Anomaly Detection in Python Using Isolation Forest | CodeSignal Learn, accessed November 7, 2025, https://codesignal.com/learn/courses/data-cleaning-and-validation-for-machine-learning/lessons/anomaly-detection-in-python-using-isolation-forest
168. Isolation Forest Guide: Explanation and Python Implementation - DataCamp, accessed November 7, 2025, https://www.datacamp.com/tutorial/isolation-forest
169. Anomaly detection using Isolation Forest - GeeksforGeeks, accessed November 7, 2025, https://www.geeksforgeeks.org/machine-learning/anomaly-detection-using-isolation-forest/
170. Isolation Forest For Anomaly Detection Made Easy & How To Tutorial - Spot Intelligence, accessed November 7, 2025, https://spotintelligence.com/2024/05/21/isolation-forest/
171. Anomaly detection in netflow traffic: workflow for dataset preparation and analysis - Frontiers, accessed November 7, 2025, https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2025.1676362/full
172. Advanced Data Anomaly Detection: Using the Power of Machine Learning - Acceldata, accessed November 7, 2025, https://www.acceldata.io/blog/advanced-data-anomaly-detection-with-machine-learning-a-step-by-step-guide

173. Anomaly Detection and Mitigation in SDN Using Machine Learning | by Dafa Saptian F, accessed November 7, 2025, https://medium.com/@dafasaptian2304/anomaly-detection-and-mitigation-for-ddos-attacks-in-sdn-using-machine-learning-1bf1bf495f56
174. Grafana dashboards | Grafana Labs, accessed November 7, 2025, https://grafana.com/grafana/dashboards/
175. Controller Runtime Controllers Detail | Grafana Labs, accessed November 7, 2025, https://grafana.com/grafana/dashboards/15920-controller-runtime-controllers-detail/
176. SDNS Dashboard | Grafana Labs, accessed November 7, 2025, https://grafana.com/grafana/dashboards/12267-sdns-dashboard/
177. Network Device Dashboard | Grafana Labs, accessed November 7, 2025, https://grafana.com/grafana/dashboards/11953-network-dashboard/
178. docker-ryu-mininet - GitHub, accessed November 7, 2025, https://github.com/iwaseyusuke/docker-ryu-mininet
179. martimy/sdn_lab: An SDN lab using Mininet and Ryu controller in Docker containers. The lab includes applications to view and create flow entries to forward packets through the network. It also includes applications to collect network metrics. - GitHub, accessed November 7, 2025, https://github.com/martimy/sdn_lab
180. jvrmaia/docker-sdn-base - GitHub, accessed November 7, 2025, https://github.com/jvrmaia/docker-sdn-base
181. stackabletech/zookeeper-operator: A tool that can be used to deploy and manager Apache ZooKeeper clusters/ensembles - GitHub, accessed November 7, 2025, https://github.com/stackabletech/zookeeper-operator
182. John-Lin/docker-ryu: SDN framework RYU in Docker - GitHub, accessed November 7, 2025, https://github.com/John-Lin/docker-ryu
183. WillFantom/Ryu-Docker: Dockerfile for Ryu SDN controller - GitHub, accessed November 7, 2025, https://github.com/WillFantom/Ryu-Docker
184. vegasbrianc/prometheus: A docker-compose stack for Prometheus monitoring - GitHub, accessed November 7, 2025, https://github.com/vegasbrianc/prometheus
185. Monitoring a Linux host with Prometheus, Node Exporter, and Docker Compose - Grafana, accessed November 7, 2025, https://grafana.com/docs/grafana-cloud/send-data/metrics/metrics-prometheus/prometheus-config-examples/docker-compose-linux/
186. How to Set Up a Monitoring Stack with Prometheus, Grafana, and Node Exporter Using Docker Compose - DEV Community, accessed November 7, 2025, https://dev.to/rafi021/how-to-set-up-a-monitoring-stack-with-prometheus-grafana-and-node-exporter-using-docker-compose-17cc
187. Building a Monitoring Stack with Prometheus, Grafana, and Alerting: A Docker Compose | by Ravi Patel | Medium, accessed November 7, 2025, https://medium.com/@ravipatel.it/building-a-monitoring-stack-with-prometheus-grafana-and-alerting-a-docker-compose-ef78127e4a19

188. On the Performance of SDN Controllers in Real World Topologies, accessed November 7, 2025, https://kic.uoi.gr/wp-content/uploads/2024/02/On_the_Performance_of_SDN_Controllers_in_Real_World_Topologies.pdf
189. [1902.04491] SDN Controllers: Benchmarking & Performance Evaluation - arXiv, accessed November 7, 2025, https://arxiv.org/abs/1902.04491
190. (PDF) SDN Controllers: Benchmarking & Performance Evaluation - ResearchGate, accessed November 7, 2025, https://www.researchgate.net/publication/331061878_SDN_Controllers_Benchmarking_Performance_Evaluation
191. Mininet Walkthrough, accessed November 7, 2025, http://mininet.org/walkthrough/
192. Performance Evaluation of SDN Controllers using Cbench and Iperf | by Disha Dudhal, accessed November 7, 2025, https://medium.com/@dishadudhal/performance-evaluation-of-sdn-controllers-using-cbench-and-iperf-e9296f63115c
193. Available online www.jsaer.com Journal of Scientific and Engineering Research, 2018, 5(9):368-375 Research Article SDN Performan, accessed November 7, 2025, https://jsaer.com/download/vol-5-iss-9-2018/JSAER2018-05-09-368-375.pdf
194. SDN-based Firewall with Ryu Controller - GitHub, accessed November 7, 2025, https://github.com/ebongemma007/SDN-Firewall
195. How to expose docker images as containernet hosts in ONOS GUI? - Stack Overflow, accessed November 7, 2025, https://stackoverflow.com/questions/72198682/how-to-expose-docker-images-as-containernet-hosts-in-onos-gui
196. Testing Ryu Multipath Routing with Load Balancing on Mininet | Wildan's Tech Blog, accessed November 7, 2025, https://wildanmsyah.wordpress.com/2018/01/21/testing-ryu-multipath-routing-with-load-balancing-on-mininet/
197. SDN Controllers: Benchmarking & Performance Evaluation - arXiv, accessed November 7, 2025, https://arxiv.org/pdf/1902.04491
198. Distributed Software-Defined Network Architecture for Smart Grid Resilience to Denial-of-Service Attacks - arXiv, accessed November 7, 2025, https://arxiv.org/pdf/2212.09990
199. SDN Performance Benchmarking: Techniques and Best Practices - ResearchGate, accessed November 7, 2025, https://www.researchgate.net/publication/381225189_SDN_Performance_Benchmarking_Techniques_and_Best_Practices/download
200. Understanding Ryu OpenFlow Controller, mininet, WireShark and tcpdump - Stack Overflow, accessed November 7, 2025, https://stackoverflow.com/questions/37998065/understanding-ryu-openflow-controller-mininet-wireshark-and-tcpdump
201. Performance Analysis of Software-Defined Networking in Band Controllers for Different Network Topologies | ADCAIJ: Advances in Distributed Computing and

Artificial Intelligence Journal, accessed November 7, 2025, https://revistas.usal.es/cinco/index.php/2255-2863/article/view/31674/30818
202. A Comprehensive Review of DDoS Detection and Mitigation in SDN Environments: Machine Learning, Deep Learning, and Federated Learning Perspectives - MDPI, accessed November 7, 2025, https://www.mdpi.com/2079-9292/14/21/4222