1. **Canary: Cloud-Native Intrusion Detection System (IDS)**

This project combines the high-performance networking of the "IDS with Honeypot" with the advanced ML-based threat detection from the "Encrypted Traffic" research project.

* **One-Sentence Description**: A high-performance, intelligent security system that detects cyberattacks in real-time by analysing network logs and encrypted traffic patterns with machine learning.
* **Real-World Use Case**: Small to medium-sized businesses often cannot afford enterprise-grade security solutions, leaving them vulnerable. Canary is a deployable security monitor that provides powerful, automated threat detection by setting up a "honeypot" (a decoy server) to attract and analyse attacks, identifying threats like brute-force attempts, vulnerability scanning, and malicious bot activity in real-time.
* **Full Tech Stack**:
  + **Core Logic & Honeypot**: **Go** is used for its high-concurrency networking to efficiently emulate services (like fake SSH/HTTP servers) and handle massive amounts of traffic.
  + **ML Analysis Engine**: A **Python** service that uses **JA3 fingerprinting** to analyse TLS handshakes from incoming traffic, feeding the data into a classifier model to identify known malicious tools (like Cobalt Strike, Metasploit) without decrypting the traffic.
  + **Data Pipeline**: Logs from the Go honeypot are streamed to **Google Cloud Pub/Sub**, which triggers a **Cloud Function** to process and store the data in **BigQuery** for analysis.
  + **Alerting & Dashboard**: A simple frontend dashboard (or a terminal UI) displays live alerts and attack statistics. Alerts can be automatically pushed to Slack or email.
  + **Deployment**: The honeypot runs on a low-cost **DigitalOcean** or **Linode** VPS to attract real-world traffic, while the analysis pipeline runs on **Google Cloud's serverless platform**.
* **Advanced Features**:
  + **Automated Threat Intelligence**: The system automatically identifies the IP addresses of attackers and adds them to a global blacklist.
  + **Geographic Attack Visualization**: A world map on the dashboard that shows the origin of attacks in real-time.
  + **Dynamic Rule Engine**: Implement a system where the IDS can be updated with new detection rules without needing a full restart.
* **How to Pitch it on a Resume**: "Developed a distributed Intrusion Detection System using Go and Python; deployed a honeypot on a cloud VPS to capture live attack data and built an ML pipeline with JA3 fingerprinting to classify encrypted threats in real-time, achieving high-accuracy detection without packet decryption.".

**STEPS**

**Step 1 : by end of 2025**

**A More Streamlined Prototyping Workflow 💡**

A more efficient path is to use Python for the *entire* prototype and introduce Go later when you need to scale.

1. **Build a Python-Only Prototype:**
   * **Simple Python Honeypot:** Quickly write a basic honeypot in Python. You don't need massive performance for a prototype. The goal is just to start collecting relevant data.
   * **Python ML Model:** Develop and train your IDS model in the same environment. This is Python's biggest strength.
   * **Benefit:** This keeps everything in a single, simple workflow. You can rapidly test your core idea: *Can my ML model detect attacks using the data my honeypot collects?*
2. **Validate and Iterate:**
   * Test the effectiveness of your data collection and ML analysis. Refine your model and the data you capture until you have a working proof-of-concept.
3. **Evolve to the Hybrid Architecture:**
   * Once your concept is proven, you can then replace the prototype Python honeypot with a high-performance **Go honeypot**.
   * Since you've already defined the data your ML model needs, you just need to make the Go service send data in the same format to your existing Python ML service. This makes the transition to a production-ready system much smoother.

**HONEYPOT**

<https://www.youtube.com/watch?v=gDjDxS55890>

Here is the distilled list of the most important modularities required.

**Required Honeypot Modules (Data Collection)**

Focus on the modules that provide the most diverse and high-value data with the least amount of setup complexity.

1. **Cloud API Honeypot:**
   * **Why it's essential:** This is the single most important module. It's what makes your project a **cloud** IDS. The data you get here (failed API calls, credential probes) is unique and cannot be captured by other honeypots.
2. **SSH Honeypot:**
   * **Why it's essential:** It's the easiest way to capture a high volume of common, automated attacks (brute-force, botnets). This provides a rich dataset for training your model to recognize baseline threats against any cloud server.
3. **HTTP/HTTPS Honeypot:**
   * **Why it's essential:** Most applications in the cloud are web-based. This module captures crucial application-layer attacks (vulnerability scans, injection attempts) that are very different from the credential attacks SSH sees.

**Most Important Infrastructure Modules (System Foundation)**

You need a minimal set of tools to make the honeypots work together.

1. **Simplified Configuration:**
   * **Why it's essential:** You need a basic way to run your honeypots (e.g., an **Argument Parser**). Don't build a complex config file system; just allow setting ports and log file paths from the command line.
2. **Standardized Logger:**
   * **Why it's essential:** This is the heart of your data pipeline for the prototype. All honeypots **must** log their data to a single location (like a JSON file) in a consistent format. This file becomes the raw dataset you will use to train and test your ML model.

**What to Ignore for the Prototype:**

* **Complex Honeypots:** Ignore Database, Container, SMB, and other specialized honeypots for now. They add complexity without proving the core concept.
* **Complex Infrastructure:** Do not build a message queue (like Kafka/RabbitMQ) or a fancy centralized logging engine. A simple "log-to-file" system is perfectly sufficient for a prototype.

**FOR SCALABILITY :**

a comprehensive list of modularities for a robust, cloud-focused Machine Learning IDS, categorized by priority.

**Tier 1: Core & Cloud-Specific Modules (Highest Priority)**

These are the absolute essentials for your project. The data from these modules will form the foundation of your cloud ML IDS.

1. **Cloud API Honeypot:**
   * **Function:** Emulates cloud provider APIs (e.g., AWS, GCP, Azure). This is your most critical module for a *cloud* IDS.
   * **Data Captured:** Attempts to use stolen credentials, privilege escalation, resource enumeration (iam:ListUsers, s3:ListBuckets), and unauthorized resource creation (launching crypto-mining VMs).
2. **SSH Honeypot:**
   * **Function:** Emulates an SSH server.
   * **Data Captured:** Brute-force/dictionary attacks, interactive command sequences from attackers, malware download attempts (wget/curl), and automated script execution.
3. **HTTP/HTTPS Honeypot:**
   * **Function:** Emulates a web server or a web application API.
   * **Data Captured:** Vulnerability scanning, SQL injection, Cross-Site Scripting (XSS), directory traversal attacks, and probes for sensitive files (/.env, /.git).
4. **Container Service Honeypot:**
   * **Function:** Emulates the Docker Engine API or a Kubernetes API server.
   * **Data Captured:** Attempts to list/run containers, misconfiguration exploits, and attempts to escape container sandboxes.

**Tier 2: High-Value Supporting Modules**

These modules expand your visibility into other common attack vectors within a typical cloud deployment.

1. **Database Honeypot:**
   * **Function:** Emulates common database services.
   * **Data Captured:** Unauthorized connection attempts, data exfiltration queries, and exploits against specific database types (e.g., Elasticsearch, Redis, MySQL).
2. **SMB/RDP Honeypot:**
   * **Function:** Emulates Windows file sharing (SMB) or Remote Desktop (RDP) services.
   * **Data Captured:** Probes for vulnerabilities like EternalBlue, ransomware propagation attempts, and brute-force login attempts on Windows-based cloud servers.
3. **Object Storage Honeypot:**
   * **Function:** Emulates a publicly accessible storage bucket (like an Amazon S3 bucket).
   * **Data Captured:** Attempts to read, write, or list files in misconfigured storage, and scans for sensitive data.

**Tier 3: Essential System & Infrastructure Modules**

These are not honeypots themselves, but are **required** to make your system functional, scalable, and manageable.

1. **Configuration Manager:**
   * **Function:** Allows you to easily enable/disable honeypots, set ports, and define logging levels. This is where your **Argument Parser** and config files (e.g., config.yaml) belong.
2. **Centralized Logging Engine:**
   * **Function:** Collects data from all active honeypot modules and standardizes it into a single format (e.g., JSON). This is crucial for feeding clean data to your ML model.
3. **Data Broker / Message Queue:**
   * **Function:** A scalable way to pass log data from your honeypots to your ML analysis engine (e.g., using RabbitMQ, Kafka, or a simple Redis queue). This is the connective tissue in a hybrid (Go/Python) architecture.
4. **Sandboxing and Isolation:**
   * **Function:** A security layer to ensure that if a honeypot is compromised, the attacker cannot break out and attack your actual infrastructure. Typically managed using Docker containers and restrictive network rules.