Comprehensive Strategy for Mission-Critical Infrastructure Optimization and Automation for Hermes Corp.

22th Oct 2024

By: Mohamad Rasoul Parsaeian

m.r.parsa@gmail.com
+989122018436

The code for this document can be found in this link:
https://github.com/mrparsaeian/ArchitectureProposal

This proposal aims to improve and automate Hermes' essential infrastructure, making sure the organization can run smoothly and reliably 24/7. The main goal is to ensure the system stays stable, scalable, and highly available as demands grow. By using automation, proactive monitoring, and secure networking, the plan looks to cut down on manual tasks, reduce downtime, and make better use of resources.

The strategy covers key areas like managing infrastructure, optimizing databases, handling messaging queues, networking, and setting up a solid disaster recovery plan. With these tools and methods in place, Hermes will have a clear path to automate resource management, keep databases and messaging systems up and running, boost network performance, and ensure strong disaster recovery.

In short, this approach will help Hermes build a reliable and efficient infrastructure that keeps its business running without any interruptions.

Automation and Optimization

The approach to automation and optimization is all about making infrastructure tasks easier by using **Ansible**, **Python** scripts, and **anomaly detection methods**. The goal is to cut down on manual work, reduce mistakes, keep configurations consistent, and proactively spot potential issues. By automating tasks like deploying, configuring, and managing servers, network devices, and virtual machines, along with using anomaly detection tools to monitor for **unusual patterns** in system metrics, the infrastructure can run more efficiently and smoothly. This helps ensure that everything stays in top shape, reducing downtime and optimizing performance.

Key Aspects:

- **ESXi Installation:** Installing and configuring **ESXi** on DL380 G10 HP servers is automated using the HP iLO REST API. This process includes creating a custom **ESXi ISO**, remotely booting and installing the servers, and setting everything up after the installation using Ansible playbooks.
- **Network Device Configuration:** Ansible automates the setup of network devices like Cisco Nexus N3K, Cisco 3750, FortiGate 60F, and datacenter switches. It handles tasks such as VLAN setup, routing, and firewall rules, ensuring consistent configuration across the network.
- **Virtual Machines Installation:** Deploying up to 200 virtual machines (VMs) on ESXi hosts is fully automated with VMware PowerCLI and Ansible. This guarantees that each VM is properly set up and ready to go.
- **Database and Messaging System Setup:** Ansible also takes care of installing and configuring tools like **RabbitMQ**, **MongoDB**, **PostgreSQL**, and **Redis**. It sets up replication for databases and clustering for message queues to ensure high availability and resilience.
- **Monitoring Tools:** System metrics are monitored with Prometheus and Grafana, while centralized log management is handled by the **ELK Stack**. Ansible scripts set up these monitoring tools, enabling proactive alerts and performance tuning.
- **Kubernetes and Docker:** Ansible automates the deployment of Kubernetes clusters with `kubeadm` and installs Docker for a consistent container runtime. This simplifies container orchestration, scaling, and redundancy for applications.

- **Anomaly Detection for Proactive Monitoring:** To boost monitoring and maintain high availability, anomaly detection is used to spot unusual patterns in system metrics. With tools like **Prophet** (a time series forecasting library from Facebook), it's possible to predict normal behavior and flag anomalies in CPU, memory, or network usage. This adds an extra layer of proactive monitoring to quickly identify and resolve potential issues.

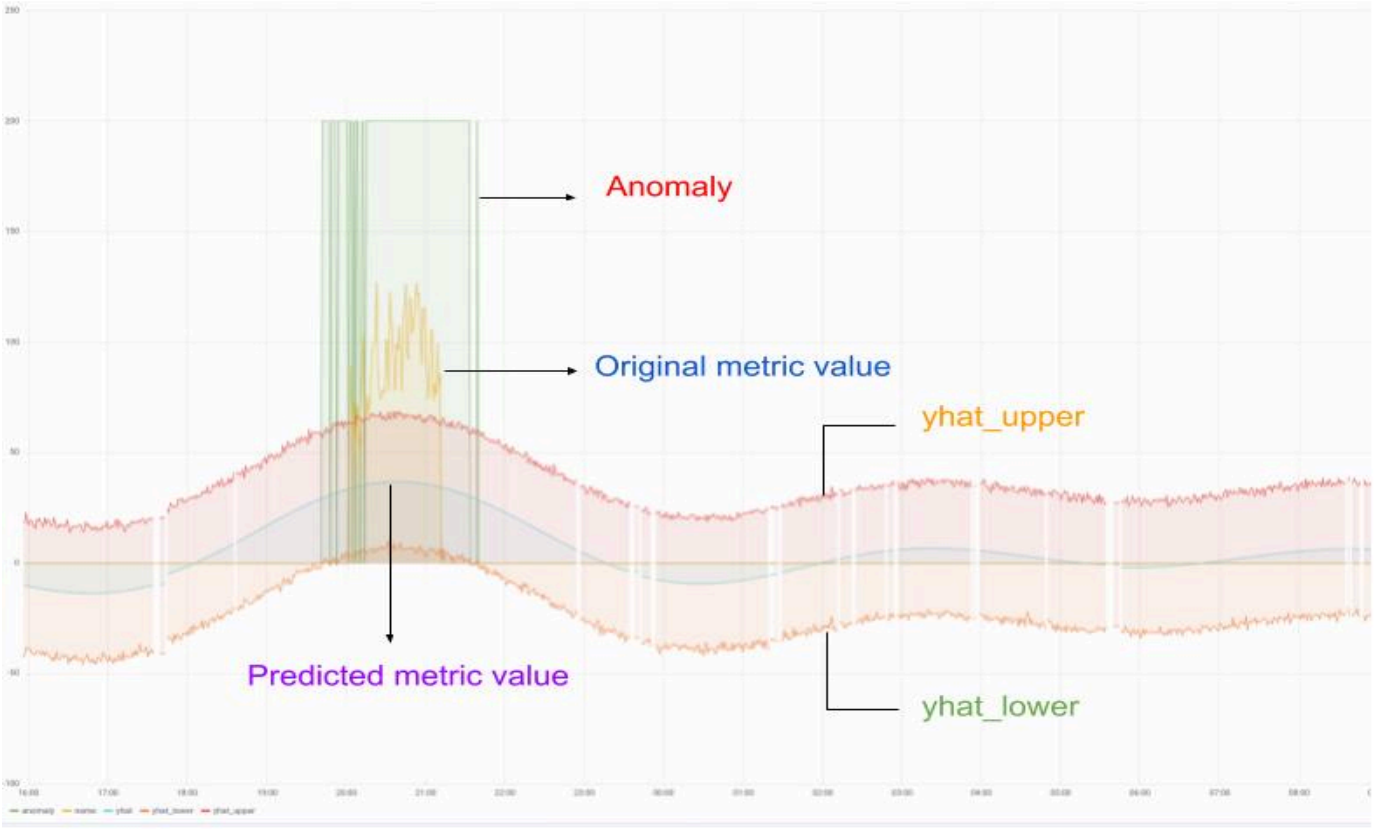
Image 1: Using istio and Prophet to generate anomaly detection graphs in grafana dashboard

Section 2: Monitoring & High-Availability

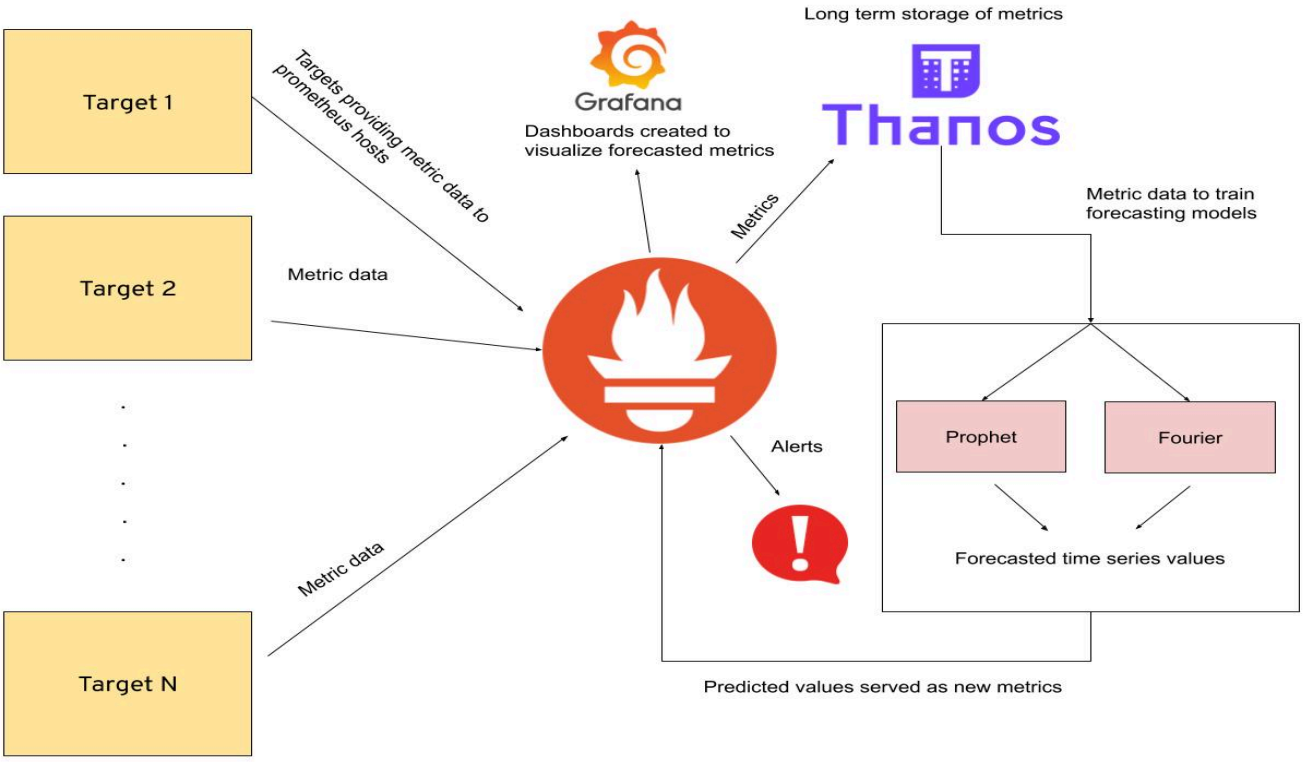
The Monitoring & High-Availability strategy is designed to keep the infrastructure running smoothly while ensuring that potential issues are detected early and addressed before they affect performance or availability. In addition to deploying robust monitoring tools, this strategy incorporates **anomaly detection methods** such as **statistical analysis**, **machine learning algorithms**, and **time series forecasting** to proactively identify **unusual patterns** or **deviations** in system metrics. By leveraging techniques like **threshold-based alerts**, **unsupervised learning** for **outlier detection**, and predictive models, the system can flag potential issues **before** they **escalate**. Along with setting up **redundancy** and enabling **automatic failover**, this approach helps minimize downtime and optimize resource usage while maintaining high availability.

Key Aspects:

- **Monitoring Tools**: **Prometheus** and **Grafana** are used for real-time monitoring of system metrics like CPU load, memory usage, and network performance. Prometheus collects data from various components, and Grafana provides visualization, allowing administrators to easily detect abnormal patterns or performance bottlenecks. These monitoring tools are configured to send alerts if performance metrics exceed predefined thresholds.
- **Anomaly Detection for Proactive Monitoring**: To enhance the monitoring and high-availability strategy, anomaly detection techniques can be incorporated to identify unusual patterns in system metrics proactively when the patterns are not detectable using traditional methods when the number of metrics is large and/or patterns are subtle . Using tools such as the **time series forecasting library Prophet**, it is possible to predict expected behavior and flag anomalies that deviate significantly from the norm. Prophet, developed by Facebook, is particularly effective for forecasting and can be integrated with existing monitoring pipelines to enhance anomaly detection. By applying time series forecasting on metrics like CPU usage, memory utilization, and network throughput, it is possible to generate alerts when anomalies occur, thereby providing an additional layer of proactive monitoring. These methods help to detect three types of anomaly, **point anomaly**, **contextual anomaly** and **collective anomaly**(link)



- **Using Istio with Prometheus and Grafana: Istio**, combined with **Prometheus** and **Grafana**, offers a powerful way to monitor and detect anomalies in Kubernetes environments by providing **real-time** insights into service mesh performance, traffic patterns, and overall system health in fast-paced changing and mission critical environment in Hermes(link).



- In addition, leveraging **Grafana and Prometheus** together with customized alert rules provides a robust mechanism for detecting anomalies in near real-time. Prometheus can collect detailed metrics, and Grafana can visualize and alert based on these metrics. Integrating anomaly alerting, as described in resources like **Grafana's alert rule customization**, allows teams to create anomaly detection alerts that can help identify issues before they escalate. This setup enhances reliability by ensuring issues are identified early, allowing for corrective action before they impact service availability.
- **Alerting System**: The **Alertmanager** tool works in tandem with Prometheus to manage and route alerts to the relevant teams. Alerts can be sent via various channels such as **email**, **Slack**, or even **SMS**. The alerting system helps administrators respond quickly when critical metrics exceed acceptable values, minimizing the risk of prolonged system issues.
- **High-Availability Setup**: To achieve high availability, **Kubernetes** is used to orchestrate workloads across multiple nodes, ensuring that services can continue running even if one or more nodes fail. By using **Kubernetes' self-healing features**, failed containers are restarted automatically, minimizing downtime. Critical databases are set up in **replication mode** to ensure data redundancy, so that if the primary node fails, a secondary node can take over without interrupting services.
- **Load Balancing**: Load balancing is implemented using **HAProxy** to distribute incoming traffic evenly across all servers. This prevents any one server from becoming overwhelmed, which not only improves the system's reliability but also boosts performance. Proper load balancing helps maintain smooth application performance even during traffic spikes.
- **Network Redundancy**: The network is designed to minimize the risk of a single point of failure by implementing **VRRP** (Virtual Router Redundancy Protocol) and **link aggregation**. These protocols enable automatic failover for network components. If a network path fails, the redundant path automatically takes over, reducing downtime and ensuring continuous connectivity.

Section 3: Database & Message Queue Management

The **Database & Message Queue Management** strategy aims to ensure data availability, consistency, and efficient message handling between microservices. This section includes practices for managing databases and message queues, with a focus on automation, replication, and performance optimization.

Key Aspects:

- **Database Management**: For managing databases like **PostgreSQL** and **MongoDB**, a combination of **replication** and **backup scheduling** is employed to ensure data is always available and recoverable. **Replication** is used for redundancy, which helps maintain data integrity even if one of the nodes goes down. Automated **daily incremental backups** and **weekly full backups** are configured to minimize manual intervention and the risk of data loss. To improve database performance, **index optimization** and **query analysis** are performed periodically to keep response times low.
- **Replication and Failover**: To prevent data loss and ensure high availability, **replication** techniques are applied across all critical databases. **PostgreSQL streaming replication**

and **MongoDB replica sets** are used for data redundancy and failover readiness. In case the primary database node fails, a secondary node is automatically promoted, ensuring minimal downtime and preventing data corruption.

- **Performance Tuning**: The performance of databases is regularly optimized through **indexing**, **query optimization**, and **parameter tuning**. Tools like **pgAdmin** are used to analyze slow-running queries and suggest improvements. Additionally, caching mechanisms are employed where needed to reduce load on databases and improve application responsiveness.
- **Backup Strategy**: **Backups** are crucial to prevent data loss. Daily incremental backups and weekly full backups are automated using **cron jobs** and **Ansible playbooks**. Regular integrity checks are also conducted to ensure the reliability of backup files, and recovery drills are performed to verify that backups are functioning as intended.
- **Message Queue Management**: To handle inter-service communication effectively, **RabbitMQ** or **Kafka** is deployed. These tools ensure that messages between services are reliably delivered, even under high load. **RabbitMQ** clustering is set up for redundancy, while **Kafka** is configured to manage a high-throughput environment. This approach ensures smooth, asynchronous communication between microservices, which is critical for a scalable architecture.

Section 4: Network & Security

The **Network & Security** strategy focuses on maintaining a secure and reliable network environment. This includes automating the configuration of network devices, ensuring redundancy for high availability, enforcing security policies, monitoring the network for suspicious activities, and securing communications between services and users.

Key Aspects:

- **Network Configuration Management**: Network configuration is automated using **Ansible** for devices like **Cisco Nexus N3K**, **Cisco 3750**, and **FortiGate 60F**. This approach helps to maintain consistency across configurations, minimizing human errors. Tasks like setting up **VLANs**, configuring routing, and updating firmware are all automated to ensure all devices remain updated and aligned with best practices.
- **Network Redundancy**: Network redundancy is achieved by implementing **VRRP** (Virtual Router Redundancy Protocol) and **HSRP** (Hot Standby Router Protocol) to provide failover capabilities for routers. This means that if one device goes down, another can take over without service disruption. Additionally, **link aggregation** is used to combine multiple physical network connections into a single logical connection, improving both redundancy and bandwidth.
- **Security Policies and Firewalls**: Security policies are enforced through firewalls like **FortiGate** and **Access Control Lists (ACLs)** to restrict unauthorized access to the network. Firewall rules are used to create network segmentation, which helps in minimizing attack surfaces by restricting unnecessary access between different parts of the network. These configurations are automated using **Ansible** to make sure all security policies are consistently enforced.

- **Network Monitoring and Intrusion Detection**: Network monitoring is essential to detect any abnormalities and potential threats. **Prometheus** and **Grafana** are used to monitor network performance metrics, such as bandwidth usage, latency, and packet loss. Additionally, **Snort** is implemented as an **Intrusion Detection System (IDS)** to detect suspicious activity on the network, such as unauthorized access attempts or unusual traffic patterns. Alerts are generated automatically if an intrusion is detected, ensuring timely intervention.
- **Secure Communication**: Secure communication protocols are implemented to protect data as it travels through the network. **TLS (Transport Layer Security)** is configured to secure data exchanges between clients and servers, ensuring data integrity and confidentiality. Additionally, **VPN (Virtual Private Network)** connections are used to establish secure tunnels between remote sites and the main network, providing an additional layer of security for sensitive communications.

Section 5: Backup & Disaster Recovery

The **Backup & Disaster Recovery** plan is focused on ensuring data protection, availability, and resilience in the face of unexpected disruptions. This section outlines how data backups are scheduled, verified for integrity, and how a disaster recovery plan is put in place, including offsite and cloud storage solutions for redundancy and snapshot management.

Key Aspects:

- **Backup Scheduling and Automation**: Backups are automated using a combination of **cron jobs** and **Ansible** to minimize manual intervention. **Daily incremental backups** and **weekly full backups** are scheduled during off-peak hours to reduce the impact on system performance. Automation helps to ensure that backups are performed consistently and that no critical data is left unprotected.
- **Data Integrity and Verification**: Ensuring data integrity is essential for reliable recovery. **Checksum** validation is used to verify the integrity of backup files. Automated scripts compare current backups against previous versions to detect any discrepancies, allowing potential issues to be identified early. Integrity checks are scheduled automatically after each backup to maintain a consistent level of data reliability.
- **Disaster Recovery Planning**: A comprehensive **Disaster Recovery Plan (DRP)** is in place to enable quick recovery of essential systems during a failure. The DRP defines **Recovery Time Objectives (RTO)** and **Recovery Point Objectives (RPO)** to establish the maximum acceptable downtime and data loss. **Disaster recovery drills** are periodically conducted to ensure the recovery process is well-practiced and efficient. Automated scripts are used to restore data and validate the recovery process.
- **Offsite and Cloud Backups**: Offsite backups are essential to prevent data loss in case of local disasters. **Abrarvan** or any other cloud provider **S3** is used as a cloud storage solution for offsite backups, ensuring data redundancy. A Python script using **boto3** is employed to upload backup files to **S3** automatically. This guarantees that a secure copy of critical data is always available in a different physical location.

- **Snapshot and Version Management**: **Snapshots** are taken regularly to create point-in-time copies of critical systems, enabling quick recovery from errors, software issues, or data corruption. These snapshots are managed to maintain **version histories**, ensuring that multiple restore points are available. **Old versions** are automatically purged based on a retention policy to optimize storage utilization while ensuring sufficient backup coverage. Snapshots for **virtual machines** and databases are created using **govc CLI** and **Ansible playbooks** to automate the process and reduce manual effort.