

INSPECKTOR GADGET : DNS/HTTP EVENT GENERATION CAPABILITIES

(RESEARCH WORK & PROPOSAL)

# **Abstract** **:**

INSPECKTOR gadgets are innovative tools designed for enhanced inspection and monitoring across various industries. These devices typically integrate advanced sensors, AI, and IoT capabilities to enable real-time data collection and analysis. Their **primary function is to detect anomalies**, **ensure quality control, and improve operational efficiency**. INSPECKTOR gadgets provide accurate diagnostics, predictive maintenance, and automated reporting. They are pivotal in reducing human error, optimizing processes, and ensuring safety and compliance standards.

Since it is an observability tool, when it works with Kubernetes, it needs access to various information related to Kubernetes objects like Pods. For that it uses **Endpoints** which is a **CRD** (Custom Resource Definition), defined in the Cilium project.

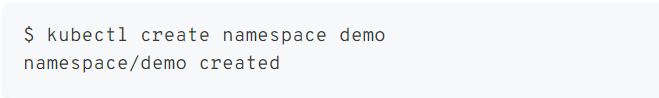
The project aims to create a CLI tool that **automates the generation of Kubernetes gadget commands for DNS tracing**. By leveraging either **Kubernetes API or external APIs**, the tool will streamline DNS event tracing with minimal manual input. The **goal** is to provide a **user-friendly interface for developers to manage DNS tracing** within Kubernetes clusters. This ensures **efficient resource management** and **real-time DNS event monitoring**.

# METHODS :

## **Using SHELL COMMANDS (***CURRENT)* :

### How It Works :

* **Namespace Creation**: Begin by creating a Kubernetes namespace where the gadget will operate.



* **Running the Gadget**: Deploy the trace\_dns gadget within the specified namespace.



* **Configuration**:The gadget starts running and listens for DNS queries and responses within the designated namespace.
* **Performing DNS Requests**: To observe DNS traffic, run a pod in a different terminal within the same namespace and perform DNS queries using tools like nslookup.



* **LOGGING & Clean up**

## **Direct Integrations With K8s API (client-go) (***SUGGESTED***):**

This method involves using the Kubernetes client libraries, such as client-go, to programmatically interact with the Kubernetes API server. The CLI tool communicates directly with the Kubernetes cluster to manage resources such as Pods or Jobs that perform DNS tracing. The tool automates the creation, management, and monitoring of these resources without requiring manual command entry.

### Requirement:

**Programming Skills:**

* Golang
* Kubernetes API

**Environment Setup:**

* Kubernetes Cluster
* Development Environment
* Kubeconfig
* Networking Knowledge:
* DNS Protocols: Understanding of DNS and how DNS tracing works within a Kubernetes cluster.

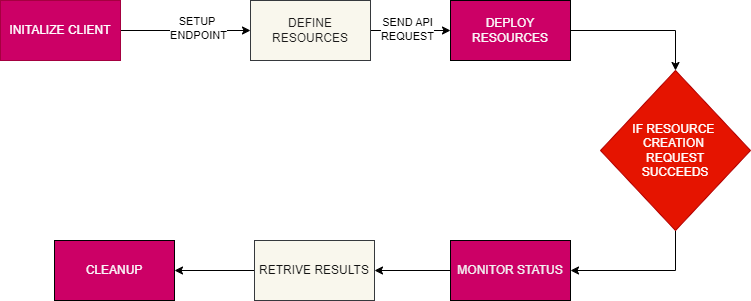
**Resource Access:**

* Cluster Permissions
* API Access.

**Tools:**

* kubectl
* Client-go library

### Workflow:



### Implementation:

* **Kubernetes Configuration:** First, ensure you have access to the Kubernetes cluster, and the kubeconfig file is correctly set up. This file allows the CLI tool to authenticate and connect to the cluster.
* **Client Initialization:** Initialize the Kubernetes client using the client-go library. This client will be responsible for sending requests to the Kubernetes API server.
* **Pod/Job Definition:** Define the Kubernetes resource (Pod or Job) that will run the DNS tracing tool. You specify the container image and other configurations like environment variables, resource limits, and volume mounts if necessary.
* **Resource Creation:** Use the client to create the defined resource in the cluster. The resource is sent to the API server, which schedules it to run on a suitable node.
* **Monitoring:** Periodically check the status of the created resource. Ensure that it transitions from the "Pending" to "Running" state, and handle any errors if the Pod fails to start.
* **Result Retrieval:**Once the tracing task is complete, access the logs or output files generated by the container. Process these results as needed.
* **Cleanup:** After retrieving the necessary data, delete the Pod or Job to free up cluster resources. Handle errors during the cleanup process if the deletion fails.
* **Error Handling:** At each stage, ensure you handle errors gracefully. For instance, if resource creation fails, provide a meaningful error message, retry if appropriate, or terminate the process cleanly.

## USING EXTERNAL API:

This method involves interacting with APIs outside the Kubernetes cluster, which may include sending HTTP requests to the Kubernetes API server or integrating with third-party services. The CLI tool uses these external APIs to initiate and manage DNS tracing tasks. This approach allows for integration with services beyond Kubernetes, offering flexibility to combine Kubernetes operations with external data sources or services.

### Requirement :

**Programming Skills:**

* Golang
* HTTP Requests

**Environment Setup:**

* API Tools
* Development Environment

**API Access:**

* External API Credentials
* API Documentation

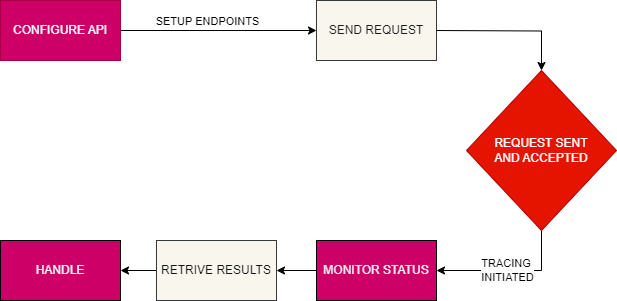
**Networking Knowledge:**

* DNS Protocols

**Permissions and Authentication:**

* API Permissions
* Authentication.

### Workflow:



### Implementation:

* **API Configuration:** Set up the necessary API endpoints and authentication credentials. This might involve configuring a token for the Kubernetes API server or integrating with a third-party service.
* **API Request Construction:** Build the HTTP request needed to initiate DNS tracing. This request typically includes the necessary payload (e.g., Pod configuration) and authentication headers.
* **Send Request:** Send the HTTP request to the relevant API endpoint. Ensure that the request is accepted and processed by the server. Handle any network issues or server errors by retrying the request or reporting the failure.
* **Progress Monitoring:** Periodically poll the API to check the status of the DNS tracing task. This ensures that the process is running smoothly or completed successfully. Handle any errors in the response, such as stalled processes or unexpected failures.
* **Result Retrieval:** Once the task is done, send another request to retrieve the results (e.g., logs or output files). Process the data as needed and ensure that the retrieval is successful.
* **Resource Cleanup:** Optionally send a DELETE request to clean up resources after the task is complete. This ensures that no unnecessary resources are left running.
* **Error Handling:** Implement error handling for each step, especially for network issues, authentication failures, and API errors. Ensure the tool provides meaningful feedback or retries operations when possible.

# # INTERNAL VS EXTERNAL METHODS:

| **Aspect** | **Direct Interaction with Kubernetes API (client-go)** | **Using External APIs** |
| --- | --- | --- |
| **Integration** | Tight integration with Kubernetes, operates entirely within the cluster. | More portable, can work across different environments. |
| **Control** | Fine-grained control over Kubernetes resources (e.g., Pods, Jobs). | Limited control, depends on the functionality of the external API. |
| **Complexity** | More complex, requires in-depth knowledge of Kubernetes and client-go. | Simpler, abstracts many Kubernetes complexities. |
| **Dependencies** | No external dependencies, everything is managed within Kubernetes. | Depends on external services and APIs, potential for downtime. |
| **Maintenance** | Requires ongoing maintenance for Kubernetes API changes and security patches. | Lower maintenance, relies on the external service provider for updates. |
| **Security** | Managed within your own infrastructure, reducing exposure to external risks. | Potential security risks with external API access and data transmission. |
| **Scalability** | Requires manual scaling and resource management within Kubernetes. | Scales easily if the external API is well-designed. |
| **Cost** | No additional costs, other than managing your own infrastructure. | May incur additional costs for using third-party services. |
| **Portability** | Tied to Kubernetes, less portable across different platforms. | More portable, can be used in non-Kubernetes environments. |
| **Use Case** | Best for Kubernetes-centric projects needing precise resource control. | Ideal for simpler, more generic solutions that need flexibility and external integrations. |

# 

# 

# **Schedule Of Deliverables**

### Milestones**:**

* **Weeks 1-6:** Establish DNS tracing functionality.
* **Weeks 7-8:** Integrate DNS tracing with testing and modularize for other gadgets.
* **Weeks 9-12:** Implement and test additional gadgets, refine and deploy the system.

## #TIMELINE (12 Weeks):

| Period | Week | Description |  |
| --- | --- | --- | --- |
| **Community & Mentor Bonding Period**  Sept 3rd - Sept 9th | 1 | Interaction & Discussion with Mentors and Community |  |
| Sept 10th - Sept 16th | 2 | * Finalizing the exact approach with mentors, With better understanding of code base and dependencies, an improved plan will be made. * Testing different approaches for PodWatcher as well as setting up environment |  |
| Sept 17th - Sept 23rd | 3 |  |
| CODING PHASE I | | |  |
| Sept 24th -Sept 30th | 4 | Setting UP Client-GO & implementation of DNS Tracing Logic |  |
| Oct 1st - Oct 7th | 5 | Implementation of PodWatcher & data collection |  |
| Oct 8th - Oct 14th | 6 | Testing, Error Handling & Optimization. |  |
| OCT 15 | MID-TERM EVALUATIONS | |  |
| Oct 16th - Oct 21st | 7 | Integration With The Testing Environment |  |
| CODING PHASE II | | |  |
| Oct 22nd - Oct 28th | 8 | Refactor and modularize the existing code to make it adaptable for other gadgets.  (common interface or abstraction layer) |  |
| Oct 29th - Nov 4th | 9 | * Research the specific requirements for the first additional gadget. * Integrate the first gadget into the modular framework. * Implement the logic to interact with the gadget and perform its specific tasks. |  |
| Nov 5th - Nov 11th | 10 | Testing & Refinement of Multi-Gadget Support |  |
| Nov 12th - Nov 18th | 11 | Documentation For the Project Implemented |  |
| Nov 19th - Nov 25th | 12 | Final Review & Deployment |  |
| Nov 26th | Final mentee evaluations and mentee feedback/blog submission. | |  |
| Nov 29th | Last Day Of term | |  |

# 

# Why me ?

I have experience with writing Go and know how Kubernetes works. These were the basic requirements to explore this project. But most of all, I am a fast learner. Before coming to this project, the Go Programming language and User perspective of K8s is all I knew.I got to explore and learn so much regarding

* Custom Resource Definition (CRDs)
* Operators
* Reconcilers
* eBPF
* Kubernetes APIs
* Kubectl