Scenic Routing

Nathan Aguilar, Julia Crumb, Megan DeYoung

Spring 2021

Senior Capstone

Table of Contents

Project Overview

General Summary of Project

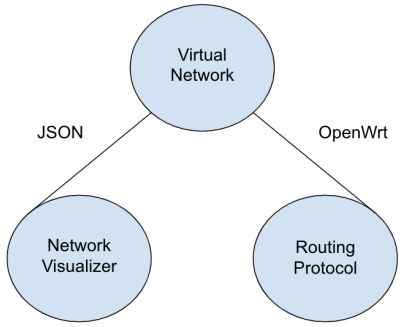
Learning about computer networking often requires access to college courses, which are not accessible to everyone who is interested in the topic. Scenic Routing is a well-organized, comprehensive, open source resource that walks computer science students through the process of creating their own computer networking projects.

The project is divided into three components:

1. Creating a virtual network of computers that can be scaled up using Kubernetes, Docker, and VrNetLab's containerized OpenWRT routers.
2. Designing and implementing a routing algorithm combining the Graph Coloring algorithm, Kruskal's Algorithm, and Breadth First Search and a routing protocol modeled after Open Shortest Path First
3. Creating a visualization of the network running the protocol to show the path of the packets of data being routed using a CentOS Virtual Machine, Unity, Elasticsearch, and Zeek.

They are connected as follows:

- The virtual network runs on a kubernete cluster. This cluster consists of multiple components encapsulated in docker containers that make up the topology of network.
- The routing protocol is configured to run on all the OpenWrt routers in the virtual network.
- The network visualizer queries the network using ElasticSearch on a CentOS 8 VM, which returns a JSON object to then display the information visually using Unity



Once developed and published, Scenic Routing will provide students with resources to create virtual networks, implement routing protocols, and visualize network traffic by providing tutorials and documentation for each component. This way, learning about computer networking can be accessible to anyone.

Motivation

Computer Networking is a subject that is often overlooked in free computer science education tools and requires a degree or certification. With the increasing importance of the internet and cloud services, it's important to have information on this topic easily available to anyone that is interested.

Existing Work

Existing Visualization Tools

Although some visualization tools for networking do exist (See Appendix C.1), very few are open-source and accessible to students. The majority are software or services sold to businesses, and the open source projects are built to show the network statically (no real-time data). There are many ways that programs can help visualize network data:

- Python has a variety of libraries and tools to graph/plot data.
- In addition, R, Java, and C/C-Sharp allow for data visualization.
- For this implementation, we chose to use Unity and C# to allow for a user-friendly interface and querying the network.

After looking at many static tools for network visualization, we found a student project from 2017 that achieves modeling network data in real time: ErgoWitness [1]. This was created as a network visualization tool, and has allowed us greater insight into the possible solutions to pulling data from our network. In this case, the project uses the ELK stack and Elasticsearch, which have been installed using Docker on a CentOS 8 Virtual Machine.

Existing Routing Protocol Projects

There are many resources on routing protocols that currently exist and how routing works. Cisco provides many comprehensive resources on routing [2, 3]. However, there are not many resources available that walk you through designing and implementing your own routing protocol. Projects that implement routing protocols are:
assigned by colleges and universities [4, 5, 8]
available on GitHub for the implementation of common routing protocols like Open Shortest Path First (OSPF) [6, 7]

The college assignments often require access to these institutions and knowledge their specific courses provide. The source code available on Github doesn’t offer any explanation or documentation to how the components of the project work or were implemented.

Existing Virtual Networking Simulators

There are many existing networking simulators out there to use as testing platforms such as EVE-NG, CORE, GNS3, and many others[##]. They all are well built for many different purposes. Although many didn’t support the openWRT router that our project relies on. Except for one pre-existing network emulator VRNetLabs[##] that containerised virtual routers.

Using VRNetLabs with Kubernetes, which is an open-source container orchestration platform[##], we can implement the network emulator using other open-source software, for our implementations we are using Meshnet-CNI[##] and k8s-topo[##].

How Are We Different?

Scenic Routing is a well-organized, comprehensive, open source resource that walks computer science students through the process of creating their own computer networking projects. Along with being well documented, each component of our project makes learning about computer networking more accessible:

- Using a virtual network simulation within docker containers to provide scalability and a testing platform that students can easily deploy.

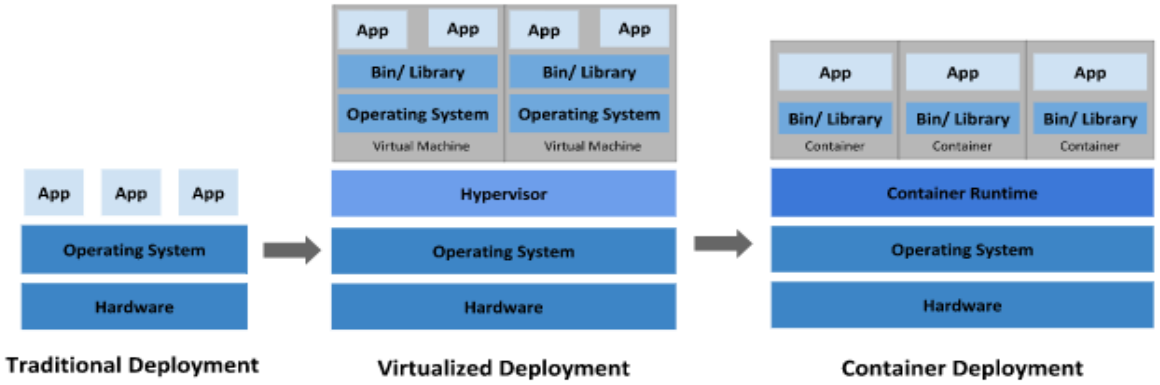
- Implementing a visualization tool that students can use to understand and evaluate the flow of information between routers and hosts on their network in real time.

- Delving into the process of creating a routing protocol to help students understand how fundamental computer science concepts like algorithms and encapsulation contribute to the way information is sent through a network.

Our work builds on existing knowledge and tools to make learning about computer networking more accessible and easy to non-experts.

Virtual Networks

Background Information



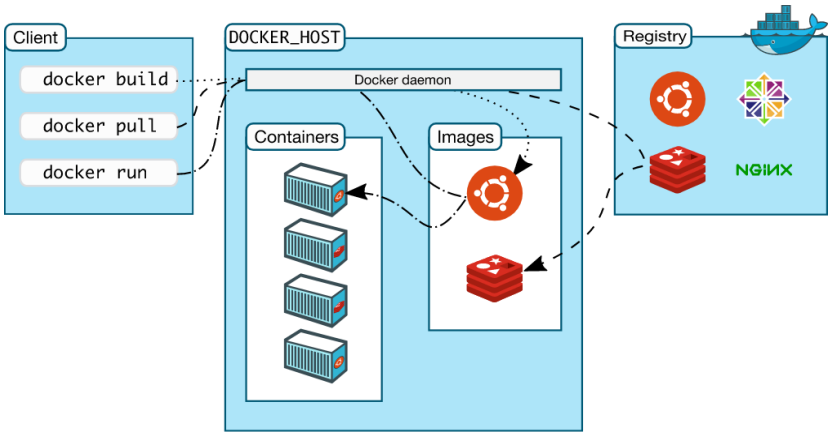
Why kubernetes for the network simulation?

Containers provide an easy way to bundle and run applications, along with kubernetes which is an open-source Container orchestration-system. Kubernetes provides a management system for containers that ensure to complete observation of the container environment. It has the functionality to scale, ensure there is no failover of your application, and provides the necessary deployment configurations, this ensures there is no downtime on your containers.[9]

Major Components

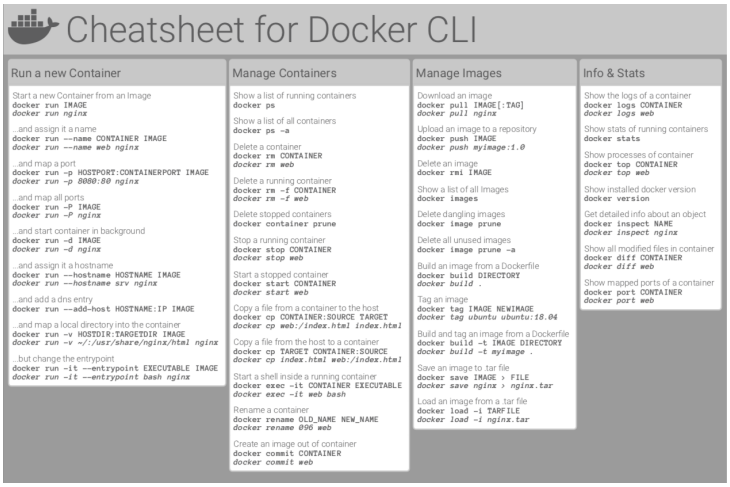
Docker

Docker is an open source containerized platform that allows the ability to quickly build, test, and deploy applications inside containers. A container represents a runtime for a single application and includes everything the software needs to run. Only sharing the OS Kernel, These isolated containers require far



less amounts of resources. The container images on the other hand are a read-on[10]

It is beneficial to try out a few commands to get yourself familiar with it. As you test on how to build different containers, it is easy to see why they have become so popular in the cloud [11]services. Containers can be quickly deployable on any one platform whether it be the server, the cloud environment or to the developer. Moreover with the integration of Kubernetes, docker is able to be easily scaled, orchestrated and deployed.[12]



VrNetlabs

VrNetlabs allows you to run different virtual routers within a containerized platform. Every router supported by vrnetlab has unique configuration and setup procedures. For the OpenWRT router, the vnetlab author created a makefile that will download the latest version of OpenWRT to the *vrnetlab/openwrt* directory and then build an OpenWRT Docker image.[13]

If you want to test the connectivity and how vrnetlab's openWRT routers work. You can follow Brianlinkletter's walkthrough[14] that shows how to implement a virtual connection between two containers. Below is an example of how I deployed two openwrt containers from the image i pulled previously and then bridged them using the using xcon, which is another container that allows the connection between two vrnetlab containers through the routers tcp interface.

```
geeknet@geeknet-MS-7971:~/vrnetlab$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
                    PORTS               NAMES
88eb21f93415        vr-xcon             "/xcon.py --p2p open…"   9 minutes ago       Exited (0)
 3 minutes ago                          vrxcon-1
f1560a0001c3        vr-xcon             "/xcon.py --p2p open…"   15 minutes ago      Exited (1)
 15 minutes ago                         bridge-openwrt1-1-openwrt-2-1
409f137dc3d8        openwrt             "/launch.py"             41 minutes ago      Exited (0)
 3 minutes ago                          openwrt2
c816e4c12fb9        openwrt             "/launch.py"             41 minutes ago      Exited (0)
 3 minutes ago                          openwrt1
geeknet@geeknet-MS-7971:~/vrnetlab$
```

```
root@geeknet-MS-7971: /home/geeknet/vrnetlab
root@geeknet-MS-7971:/home/geeknet/vrnetlab# docker logs vrxcon-1
2020-11-16 12:01:12,075: xcon       DEBUG    00102 bytes openwrt1/1 -> openwrt2/1
2020-11-16 12:01:12,075: xcon       DEBUG    00102 bytes openwrt2/1 -> openwrt1/1
2020-11-16 12:01:13,075: xcon       DEBUG    00102 bytes openwrt1/1 -> openwrt2/1
2020-11-16 12:01:13,076: xcon       DEBUG    00102 bytes openwrt2/1 -> openwrt1/1
2020-11-16 12:01:14,075: xcon       DEBUG    00102 bytes openwrt1/1 -> openwrt2/1
2020-11-16 12:01:14,076: xcon       DEBUG    00102 bytes openwrt2/1 -> openwrt1/1
2020-11-16 12:01:15,075: xcon       DEBUG    00102 bytes openwrt1/1 -> openwrt2/1
2020-11-16 12:01:15,076: xcon       DEBUG    00102 bytes openwrt2/1 -> openwrt1/1
```

Kubernetes

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.[13] Kubernetes deploys a control plane when you initialize the cluster, along with any workers nodes you may deploy with the control plane specified in your configuration file. Kubernetes is a fantastic orchestration tool that allows for testing out many networking provisions. Starting with the major components of the cluster will be the control plane, these components are stored inside the control plane as pods.

Control Plane Components

kube-apiserver

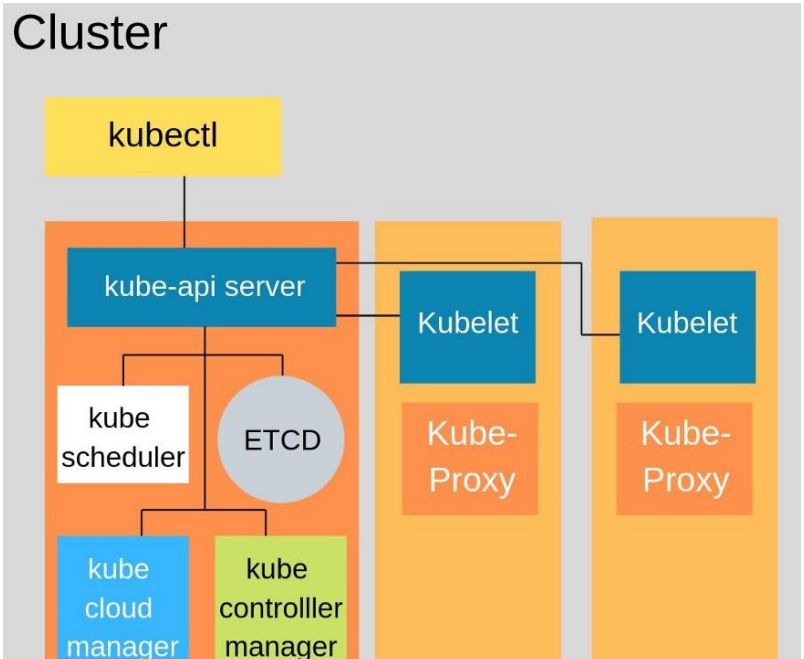
The Kubernetes API server validates and configures data for the api objects which include pods, services, replicationcontrollers, and others. The API Server services REST operations and provides the frontend to the cluster's shared state through which all other components interact.

Kube-proxy

The Kubernetes network proxy runs on each node. This reflects services as defined in the Kubernetes API on each node and can do simple TCP,UDP stream forwarding or round robin TCP,UDP forwarding across a set of backends. Service cluster ips and ports are currently found through Docker-links-compatible environment variables specifying ports opened by the service proxy. There is an optional addon that provides cluster DNS for these cluster IPs. The user must create a service with the apiserver API to configure the proxy.

Kube-scheduler

The Kubernetes scheduler is a policy-rich, topology-aware, workload-specific function that significantly impacts availability, performance, and capacity. The scheduler needs to take into account individual and collective resource requirements, quality of service requirements, hardware/software/policy



constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, deadlines, and so on. Workload-specific requirements will be exposed through the API as necessary.

kubelet

The kubelet is the primary “node agent” that runs on each node. The kubelet works in terms of a PodSpec. A PodSpec is a YAML or JSON object that describes a pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) and ensures that the containers described in those PodSpecs are running and healthy.

kube-controller-manager

The Kubernetes controller manager is a daemon that embeds the core control loops shipped with Kubernetes. In applications of robotics and automation, a control loop is a non-terminating loop that regulates the state of the system. In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired state.[15]

Etcd

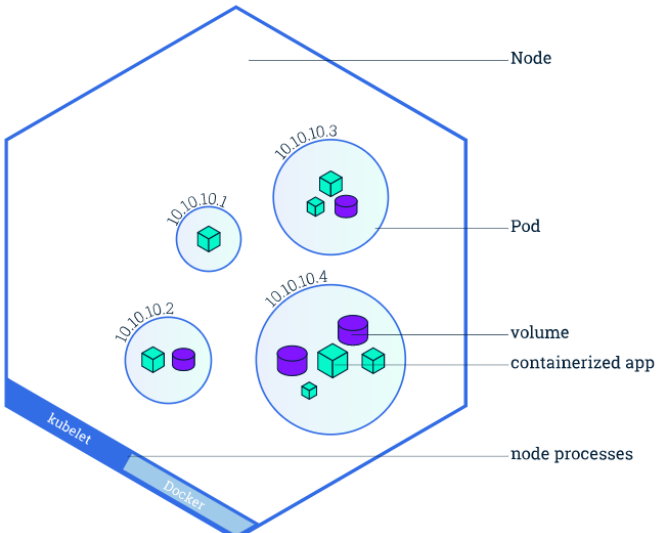
etcd is a consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

cloud-controller-manager

A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

Getting to know your node

Within a cluster you can have numerous nodes, with one being a master, the rest being worker nodes, for the network simulation we will only be working with a single node because we aren't distributing this on a cloud service. Rather we are trying to test the topology and understand the network interface within a single node cluster. Thus we need the



control plane(master node), and k8s-topo, this will be a kubernetes add-on we talk about later on, to build our network simulation through pods.
Lastly our final piece of the puzzle would be connecting our pods within our master node through meshnet-CNI which will deploy the virtual connections between the pods based on k8s-topo topology.

Kubernete Objects

In order to provide kubernetes with an object you must provide k8s(kubernetes) with object specs that tell the orchestration system the information needed in order to create the object.
In order to create the desired object such as our meshnet-CNI or k8s-topo we need to send the kubernetes API some yaml file(these files are kubernetes main file input) with at most the required information:

- `apiVersion` - Which version of the Kubernetes API you're using to create this object
- `kind` - What kind of object you want to create
- `metadata` - Data that helps uniquely identify the object, including a `name` string, `UID`, and optional `namespace`
- `spec` - What state you desire for the object

```yaml
---
apiVersion: v1
kind: List
items:
- apiVersion: networkop.co.uk/v1beta1
  kind: Topology
  metadata:
    name: r1
  spec:
    links:
    - uid: 1
      peer_pod: r2
      local_intf: eth1
      peer_intf: eth1
      local_ip: 12.12.12.1/24
      peer_ip: 12.12.12.2/24
    - uid: 2
      peer_pod: r3
      local_intf: eth2
      peer_intf: eth1
      local_ip: 13.13.13.1/24
      peer_ip: 13.13.13.3/24
- apiVersion: networkop.co.uk/v1beta1
  kind: Topology
  metadata:
    name: r2
  spec:
    links:
    - uid: 1
      peer_pod: r1
      local_intf: eth1
      peer_intf: eth1
      local_ip: 12.12.12.2/24
      peer_ip: 12.12.12.1/24
    - uid: 3
      peer_pod: r3
      local_intf: eth2
      peer_intf: eth2
      local_ip: 23.23.23.2/24
```

Kubectl takes the yaml file and sends it to the API to process the data. At which point the control plane will begin to implement the data you had specified.

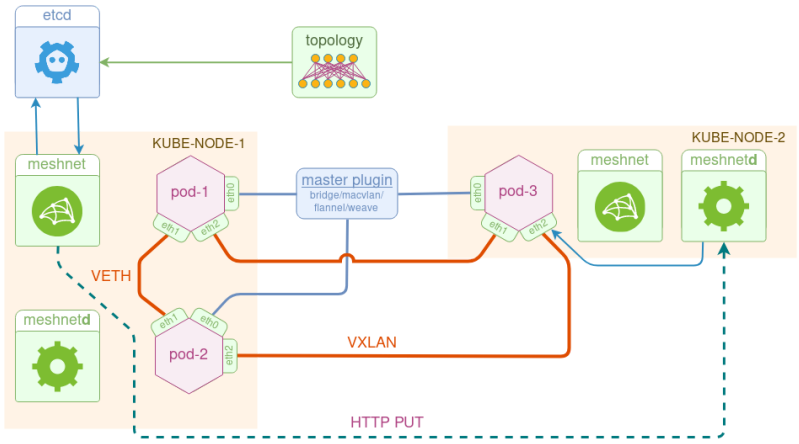
Why is disabling swap off important?

Support for swap is non-trivial. Guaranteed pods should never require swap. Burstable pods should have their requests met without requiring swap. BestEffort pods have no guarantee. The kubelet right now lacks the smarts to provide the right amount of predictable behavior here across pods.As of today, the scheduler does not understand swap configuration and usage to provide smarts for swap support.[16]

Meshnet-CNI

A CNI, Container Network Interface, is used to orchestrate the network between pods. It takes a container runtime and connects them to a network. This allows for the pods of our topology builder to communicate with one another through a virtual connection.[17]

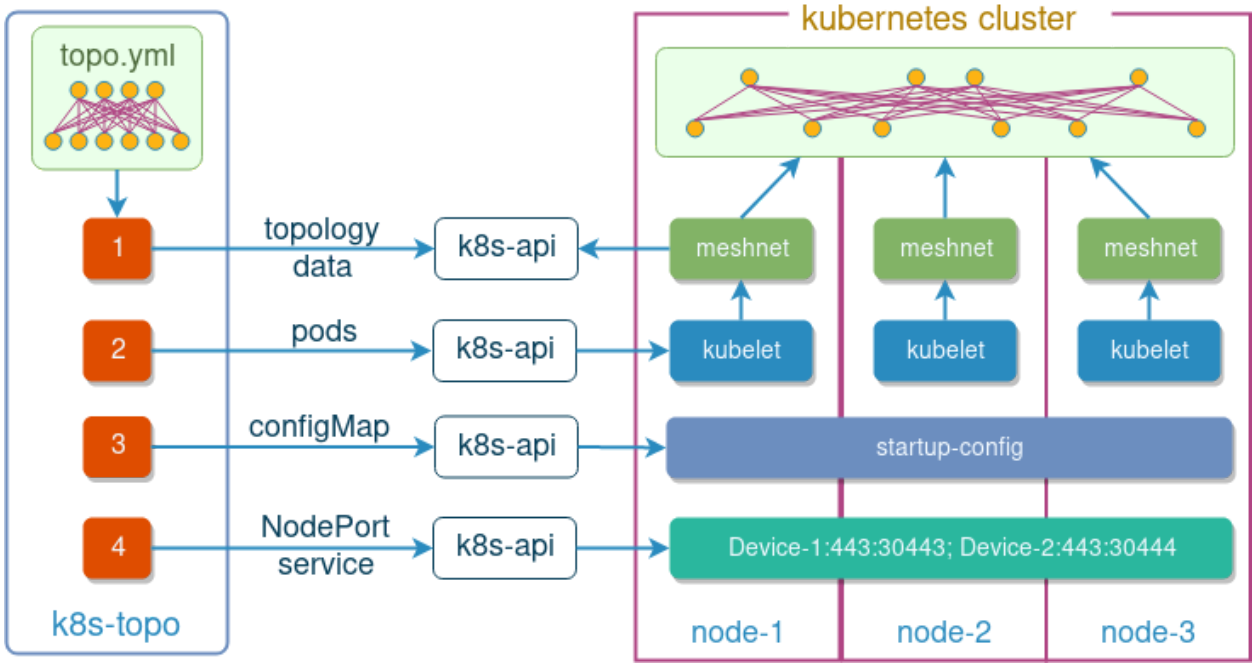
For us this will be meshnet-CNI which creates arbitrary network topologies out of point-to-point links with the help of koko.[18] Interconnects these pods through the defined topology. In the images below I applied a topology defined by the three node yaml file. Which deployed three additional pods to my cluster.




```
geeknet@geeknet-MS-7971:~/meshnet-cni$ kubectl apply -f tests/3node.yml
topology.networkop.co.uk/r1 created
topology.networkop.co.uk/r2 created
topology.networkop.co.uk/r3 created
pod/r1 created
pod/r2 created
pod/r3 created
geeknet@geeknet-MS-7971:~/meshnet-cni$ kubectl get pods -l test=3node
NAME   READY   STATUS    RESTARTS   AGE
r1     0/1     Pending   0          9s
r2     0/1     Pending   0          9s
```

Next we pinged the pods to verify that the connectivity between them was established.

```
geeknet@geeknet-MS-7971:~/meshnet-cni$ kubectl get pods -l test=3node
NAME   READY   STATUS    RESTARTS   AGE
r1     1/1     Running   0          5m45s
r2     1/1     Running   0          5m45s
r3     1/1     Running   0          5m45s
geeknet@geeknet-MS-7971:~/meshnet-cni$ kubectl exec r1 -- ping -c 1 12.12.12.2
PING 12.12.12.2 (12.12.12.2): 56 data bytes
64 bytes from 12.12.12.2: seq=0 ttl=64 time=0.054 ms

--- 12.12.12.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.054/0.054/0.054 ms
geeknet@geeknet-MS-7971:~/meshnet-cni$ kubectl exec r2 -- ping -c 1 23.23.23.3
PING 23.23.23.3 (23.23.23.3): 56 data bytes
64 bytes from 23.23.23.3: seq=0 ttl=64 time=0.059 ms

--- 23.23.23.3 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.059/0.059/0.059 ms
geeknet@geeknet-MS-7971:~/meshnet-cni$ kubectl exec r3 -- ping -c 1 13.13.13.1
PING 13.13.13.1 (13.13.13.1): 56 data bytes
64 bytes from 13.13.13.1: seq=0 ttl=64 time=0.052 ms
```



K8s-topo[19]

K8s-topo provides our CNI with the topology to build our network. It uses an arbitrary network topology builder for network simulations without our cluster. This also ties into our vrnetlabs container image, as we can use that image to create a virtual openwrt router within a pod. Once we have a topology defined with the router we can begin to test different configurations.

Routing Protocol

Background Information

To fully understand the completed components of the Routing Protocol portion of Scenic Routing here is some basic information about routing, graph theory, and algorithms.

Routing

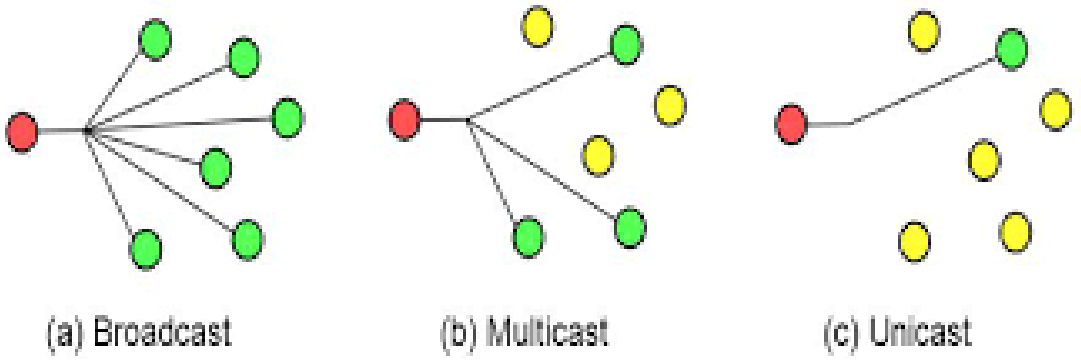
In computer science, routing refers to the process of learning all the routes, or paths through a network, and using them to forward data from one network to another or within the same network. [20] This data is made up of a packet or several packets.

Packets can be routed through broadcast, multicast, or unicast.

a) **Broadcast** (One to All) - a single of a packet sent by a single sender to the broadcast address, which is listened to by all hosts in the network.

b) **Multicast** (One to Many) - a single packet sent by a single sender to the multicast address, which is only listened to by hosts in the multicast group.

c) **Unicast** (One to One) - a single packet sent by single sender to a single recipient.



[21]

A routing protocol is a set of processes, algorithms, and messages that are used to exchange routing information to determine the paths in the network, often the best paths. Routing protocols are used to facilitate the exchange of routing information between routers. [22]

A routing protocol will build and update a routing table. A Routing table is a data table stored in a router or a network host that lists the routes to particular network destinations, and in some cases, metrics associated with those routes.

The basic components of a routing protocol are as follows. The bolded components on the list are what has been implemented thus far in Scenic Routing. [22]

- **Routing Algorithm** - Determines the path from one network node (router or host) to another.
- **Hello Protocol** - Ensures that the communication between routers is bidirectional by periodically sending packets to neighboring routers, then updating the list of router neighbors depending on if the packets it sends are acknowledged.
- Flooding Protocol - Updates all routers in the network with information about changes to each router's neighbor list until all routers have the same information.
- Calculating Metrics - Determines how to calculate how expensive it is to forward packets from one router to another using throughput, bandwidth, reliability, etc.
- Routing Table Creation - Uses the neighbor list information, metrics, and routing algorithm to create a routing table.

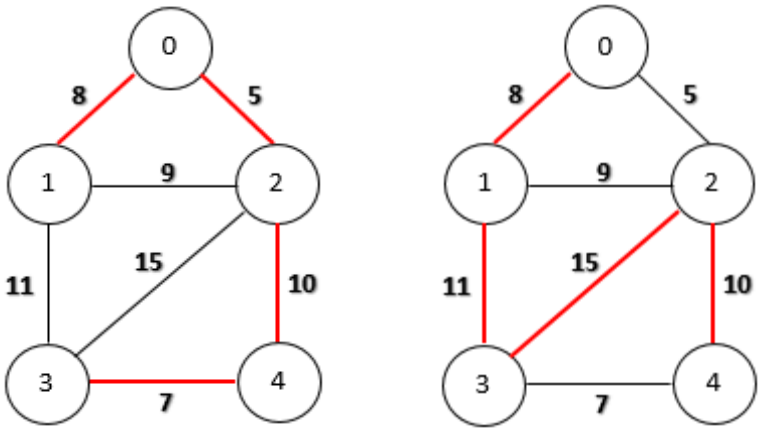
Graph Theory and Algorithms.

These are the graph theory concepts and algorithms used in the implementation of the Scenic Routing protocol's routing algorithm.

Graph Theory Concepts

A computer network can be represented as a graph. To be more specific, a computer network can be represented by an undirected, cyclic graph. This means that all of the edges between nodes are bidirectional and it is possible that the graph has loops.

A minimum spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the nodes together, has no cycles, and uses the minimum possible total edge weight. A maximum spanning tree is the same except it uses the maximum possible total edge weight. Below you will find the minimum spanning tree (left) and maximum spanning tree (right) for the same graph. [23]



[24. 25]

Kruskal's Algorithm

Kruskal's algorithm is a greedy algorithm that finds a minimum spanning tree of an undirected edge-weighted graph. This algorithm works as follows:

1. Sort all the edges from low weight to high
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

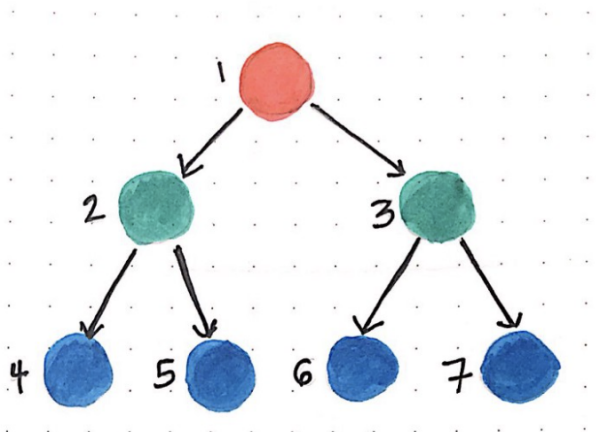
[26]

Breadth First Search

Breadth First Search is an algorithm used to search or traverse a graph. It does this by starting from a root node (which can be any node in the graph) and exploring all of the neighbor nodes at its present depth before moving onto the next level of nodes. The algorithm stops when all of the nodes in the graph have been visited. This can be modified to stop when a specific node has been visited if looking to find a path between two nodes.

[27]

In the diagram below, the Breadth First Search algorithm would visit all of the nodes in the orange level, then the green level, and finally the blue level.



[28]

Graph Coloring Method

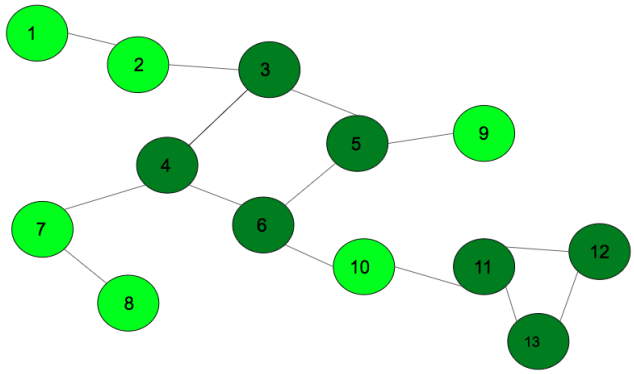
The Graph Coloring Method is used to find all the cycles in a simple, undirected graph. Coloring refers to marking the nodes based on if they are visited, partially visited, or unvisited using black, gray, and white respectively.

The algorithm works as follows:

1. Call Depth First Search traversal for the graph which can color the nodes.
2. If a partially visited node is found, backtrack till the node is reached again and mark all nodes in the path with a counter which is cycle number.
3. After completion of traversal, iterate for cyclic edges and push them into a separate adjacency list.
4. Print the cycles number wise from the adjacency list.

[29][30]

This image below uses the Graph Coloring method to detect cycles, the cycles noted in dark green.



What is the Scenic Routing Protocol?

The Scenic Routing routing protocol will be a link-state, Interior Gateway Protocol (IGP) written using C/C++. This means that the protocol program will run on every router within a closed, isolated network and dynamically change routes based on updates in the network. The protocol's algorithm will aim to route the packets in the longest possible path to their destination while avoiding infinite loops.

Finished Routing Protocol Components

As mentioned before, the completed, fully documented components of the routing protocol includes the Routing Algorithm and the Hello Protocol.

Routing Algorithm

As previously mentioned, the goal of the Scenic Routing routing protocol is to route packets the longest route possible through the network. This path will be determined by the routing algorithm. However, finding the longest path is not as simple as finding the shortest.

In theoretical computer science, the longest path problem is the problem of finding a simple path of maximum length in a given graph, simple meaning that the path does not have any repeated nodes. The length of a path is measured by either the number of its edges, or by the total weight of its edges This is otherwise known as the "Travelling Salesman Problem" or the "Hamiltonian Cycle Problem" [31]

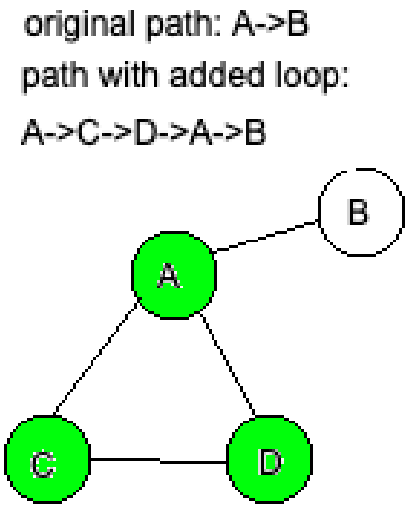
The longest path can be found of a graph that is directed and acyclic (has no cycles), but a computer network is not directed or strictly acyclic.

For this reason, our algorithm will not be truly the longest but "longish".

Design

The design of the routing algorithm is a combination of Kruskal's algorithm, Breadth First Search, and the Graph Coloring Method. By negating the edge weights of the graph, Kruskal's algorithm will produce a maximum spanning tree. Breadth First Search will then be used to find the path from one node on the maximum spanning tree to another. This ensures that the path between the two nodes will be made up of the largest edge weights possible, therefore potentially having a larger total weight.

This design so far still leaves room for our "longish path" to be the same as the shortest path. For example, if the two nodes are adjacent to each other, even after the original graph is transformed into a maximum spanning tree, then both the longest and shortest path will be the same. This can be avoided if a loop is attached to the front of the path that includes the source node, but does not include the destination node. Take the following diagram for example:



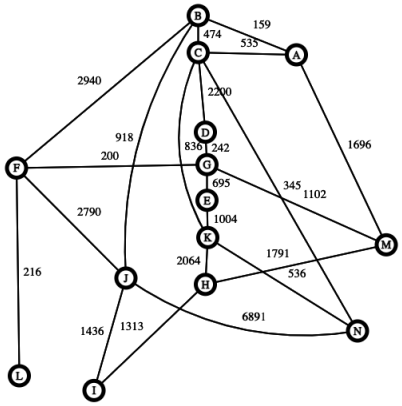
This is achieved by using the Graph Coloring method to find all the loops in the original graph then storing them to be accessed later if the nodes are adjacent to one another.

Implementation

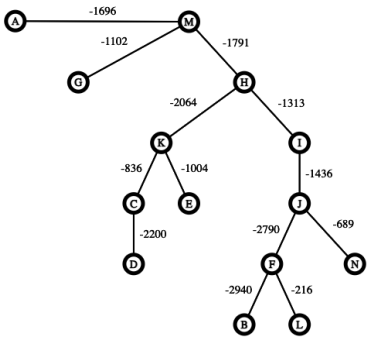
The routing algorithm was implemented in C++. The program takes in a file with each node, edge, and edge weight needed to build the original graph. The algorithm is then applied and the user is able to enter a source and destination node. After printing out each edge in the determined path, the total cost of the path is displayed.

The following images are an example of the original graph (A), corresponding maximum spanning tree (B), and user input and program output (C) for a path with a non-adjacent source and destination.

A



B



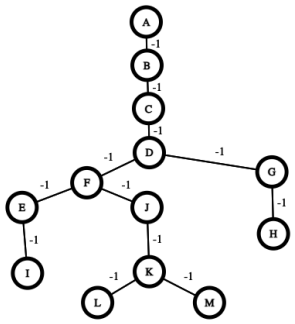
C

```
-----------------------------------
SCENIC ROUTING: longest path finder
-----------------------------------

Enter the name of the source node: K
Enter the name of the destination node: N

K -> H  Cost: 2064
H -> I  Cost: 1313
I -> J  Cost: 1436
J -> N  Cost: 689

TOTAL COST: 5502


Press any key to continue OR q to QUIT
```

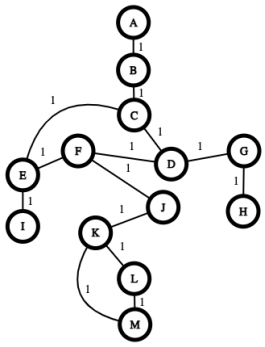
In the original graph, the nodes K and N are adjacent to each other, so the shortest path is a single edge with a total weight of 536. After the algorithm is applied to the original graph, the new path found is KHIJN for a total weight of 5502, which is much longer.

The next set of images is an example of the original graph (D), corresponding maximum spanning tree (E), and user input and program output (F) for a path with an adjacent source and destination that can have the loop added on.

D



E



F

```
-----------------------------------
SCENIC ROUTING: longest path finder
-----------------------------------

Enter the name of the source node: K
Enter the name of the destination node: J

K -> M  Cost: 1
M -> L  Cost: 1
L -> K  Cost: 1
K -> J  Cost: 1

TOTAL COST: 4


Press any key to continue OR q to QUIT
```

In the original graph, the nodes K and J are adjacent to each other and remain adjacent to each other in the maximum spanning tree. So, the path in the original graph is a single edge with a total weight of 1. After the algorithm is applied to the original graph, the new path found includes the loop MLK, making the new path KMLKJ with a weight of 4.

Hello Protocol

What is a Hello Protocol?

The purpose of a Hello Protocol is to establish and maintain neighbor relationships. It also ensures that communication between neighbors is bidirectional. [32]

Hello packets are sent periodically out all router interfaces. When a neighboring router receives a hello packet, it sends a new hello packet as an acknowledgement. Bidirectional communication is indicated when the router sees itself listed in the neighbor's Hello Packet. [33]

For the purposes of this project, the network is a broadcast network, meaning that multicasting and UDP can be used in the Scenic Routing Hello Protocol.

Design

Scenic Routing implements a much simpler Hello Protocol than normally used in other routing protocols. This is because neighbor discovery is not being implemented, rather a file containing

the nodes and edges in the network is read from on each router and is used to determine neighbors. The hello protocol is being used to check if the router's neighbors are still up and able to communicate with.

The hello packet contains:

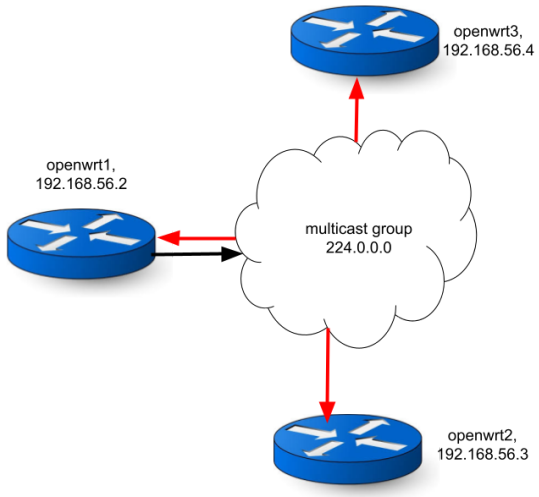
- The IP address of the router sending the hello
- The IP address of either the destination router or the multicast address.
- The time the packet was constructed
- A character to represent whether the packet was an acknowledgment or not.

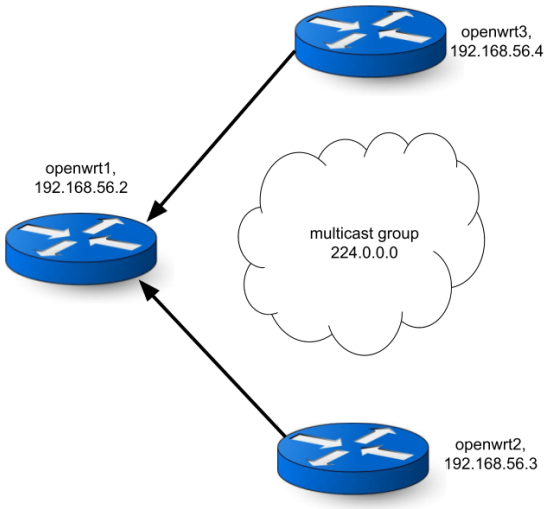
So, instead of the router waiting to see itself in another hello packet's list of neighbors, the router keeps track of the time it last received an acknowledgement from one of its neighbors. If a certain amount of time has passed before receiving acknowledgement, the router is marked as down. Alternately, if a router suddenly receives an acknowledgment from a previously down router, it is marked as a neighbor again.

Implementation

The Hello Protocol was implemented in a combination of C and C++ and then cross-compiled to run on OpenWrt routers. The program itself utilizes the node and graph classes from the routing algorithm program to build the original list of neighbors. The program takes a single argument, its own IP address.

Threads are used to implement the periodic sending of hello packets to the multicast address, listening on the multicast address to receive multicast packets then send acknowledgements, and receive acknowledgement packets. This is to ensure that each of these tasks can happen concurrently and have access to the container that holds the previous time of acknowledgement for each of the router's neighbors.





The diagrams above depict a router sending a hello packet to the multicast address and having the routers in the multicast receive that packet and the acknowledgements being sent back to that router.

The program currently sends and receives all hello packets, but the updating of the acknowledgements has yet to be implemented. In the future the program will also be able to find its own IP address without needing an argument.

Open Source Documentation

Here is the current list of topics covered by the documentation on GitHub:

- Routing Basics
- OpenWrt Router Build System Setup
- OpenWrt Cross Compilation Guide
- OpenWrt Router Setup Through VirtualBox
- Longest Path Algorithm Design
- Longest Path Algorithm Pseudocode
- Longest Path Algorithm Project Guide
- Hello Protocol Basics
- Hello Protocol Project Guide

Technologies Used

C/C++

The code for both the Routing Algorithm and Hello Protocol was written in a combination of C and C++ in the IDE Eclipse and Geany.

OpenWrt

OpenWrt (open wireless router) is an open-source project for embedded operating systems based on Linux, primarily used on embedded devices to route network traffic. The main components are Linux, util-linux, musl, and BusyBox. [34][35]

In this project, OpenWrt is used as a router that can install user-defined firmware packages. This allows the user to write their own code and run it on a router to act as an interface.

OpenWrt Build system

The OpenWrt build system is a set of Makefiles and patches that automates the process of building a cross-compilation toolchain and then using it to build pieces of software to run OpenWrt on a specific device or software to run on an OpenWrt router. A typical toolchain consists of:

- a compiler, such as gcc
- binary utilities such as an assembler and a linker; for example binutils
- a C standard library, such as glibc, musl, uClibc or dietlibc

For this project, the OpenWrt Build System is used to cross-compile source code from a development machine or environment to an .ipk file (software) that can be installed and run on the OpenWrt Router.

[36]

Virtual Box

For this project VirtualBox was used to create virtual machines to run the x86-64 OpenWrt images and an OpenSUSE Linux development machine.

Network Visualizer

Background Information

The network visualizer will provide an interface for users to view network data in real time, using Unity and C#. The interface will query the network to determine the flow of packets, and render that data with 3D models and animations. This implementation is based on ErgoWitness, a 2017 project which uses Elasticsearch to record and send network data, and a Unity engine interface to display the data in real time.

Technologies Used

The final implementation of the visualization will use many technologies and tools, including:

- Elasticsearch
- Unity
- CentOS
- Docker
- Zeek

Elasticsearch

Excerpt from Appendix []-Notes on Intellipaat Elasticsearch Tutorial:
What is ELK?

Elastic Stack is a group of open source products from Elastic (company) designed to help users take data from any type of source and in any format and search, analyze, and visualize that data in real time.

The heart of Elastic Stack is Elasticsearch, which drives everything.

Elastic Stack presents a steeper learning curve than some comparable products (you need to set up everything), as well as more set up, owing in part to its open source nature. In return for the extra work, however, the sysadmin is rewarded with a deeper understanding of the software's underlying structure.

Elastic was founded in Amsterdam in 2012 to support the development of Elasticsearch and related commercial products and services.

Elasticsearch, Logstash, Kibana are the three main components

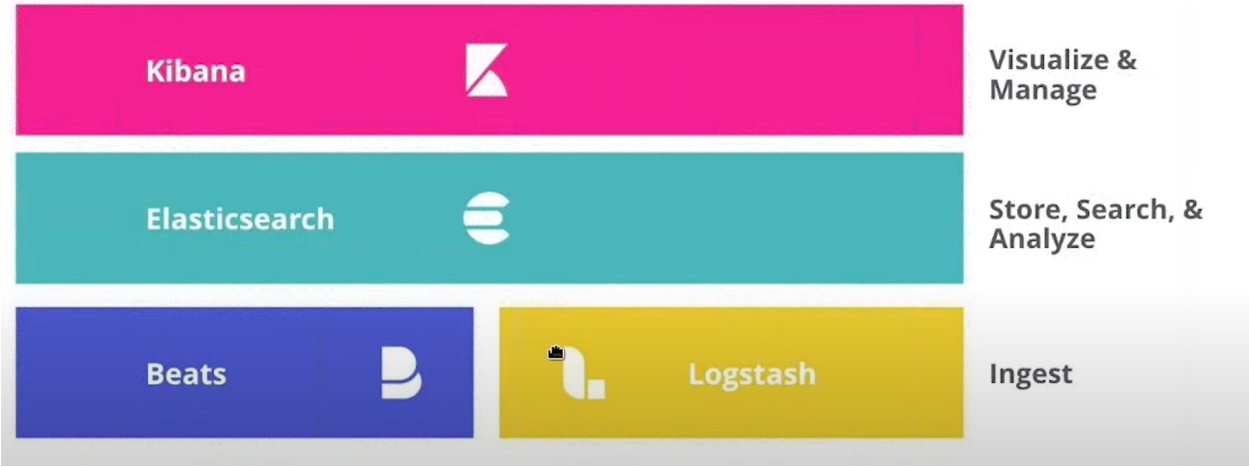
Components of Elastic Stack

Elasticsearch: We can store data in Elasticsearch

Logstash: A data pipeline that we can use to read data from various sources, and can write it to various sources. It also provides a feature to filter the input data before sending it to output

Kibana: A graphical user interface that we can use to do a lot of things

Beats: Lightweight data shippers that sit on different servers and send data to Elasticsearch directly or via Logstash



ELK Stack to Elastic Stack

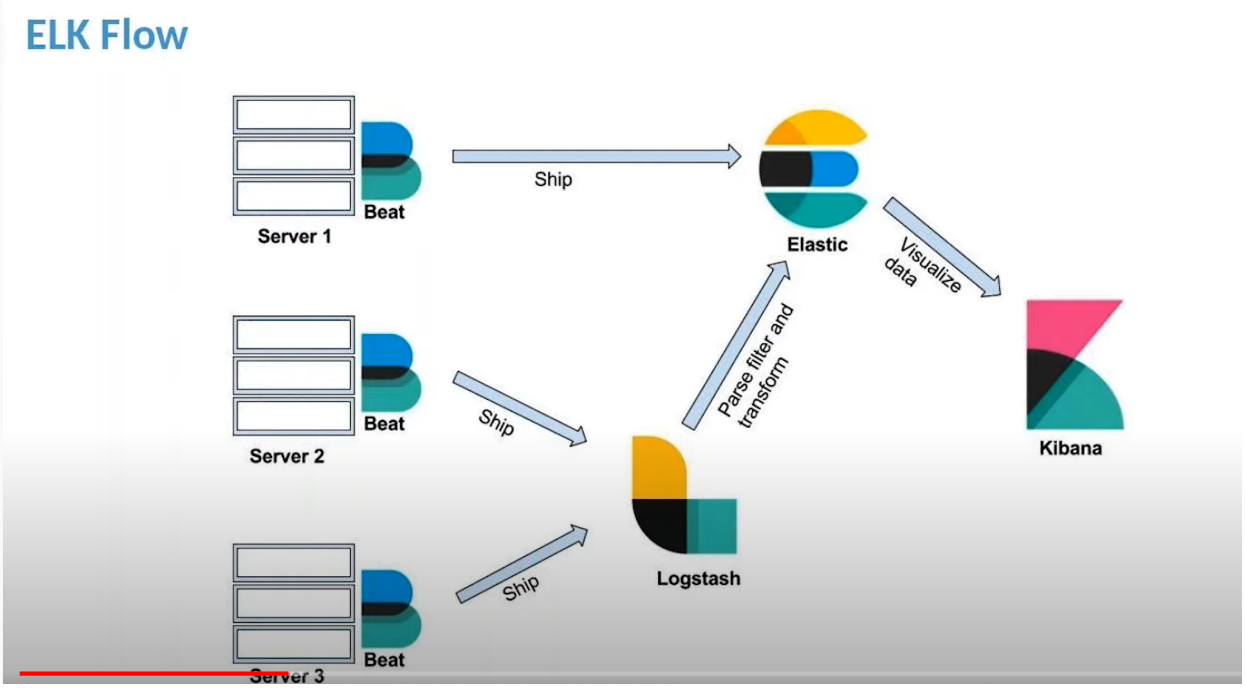
It all started with Elasticsearch:

The open source, distributed, RESTful, JSON-based search engine. Easy to use, scalable, and flexible

Then it grew with Logstash and Kibana

Later as the community expanded and the use cases increased. They now added Beats on ELK, so this brought one additional tool.

The Elastic Stack is the ELK Stack, but with more flexibility to do great things.



Elasticsearch

Elasticsearch is a full-text search engine that's primarily used for searching. It can also be used as a NoSQL database and analytics engine.

Elasticsearch is basically schema-less and works in near-real time.

It has a RESTful interface, which helps us to interact with it easily from multiple interfaces.

Elasticsearch supports different ways of importing various types of structured or unstructured data; it handles all types of data because of its schema-less behavior and it's quite easy to scale.

Kibana

In Elastic Stack, Kibana is mainly used to provide the graphical user interface, which we use to do multiple things.

When Kibana was first released, we just used it to create charts and histograms, but with each update, Kibana evolves and now we have lots of killer features that make Kibana stand out from the crowd.

There are many features in Kibana, but when we talk about the key features,

1. Discover your data by exploring it
2. Analyze your data by applying different metrics
3. Visualize your data by creating different types of charts
4. Apply machine learning on your data to get data anomaly and future trends
5. Monitor your application using APM (Application Performance Monitoring)

6. Manage users and roles
7. Run Elasticsearch expressions with a built-in console
8. Play with time-series data using Timeline
9. Monitor your Elastic Stack using Monitoring

Logstash

Logstash is a data pipeline that can take data input from various sources, filter it, and output it to various sources; these sources can be files, Kafka, or databases.

Logstash is a very important tool in Elastic Stack as it's primarily used to pull data from various sources and push it to Elasticsearch; from there, Kibana can use that data for analysis or visualization.

We can take any type of data using Logstash, such as structured or unstructured data, which comes from various sources, such as the internet.

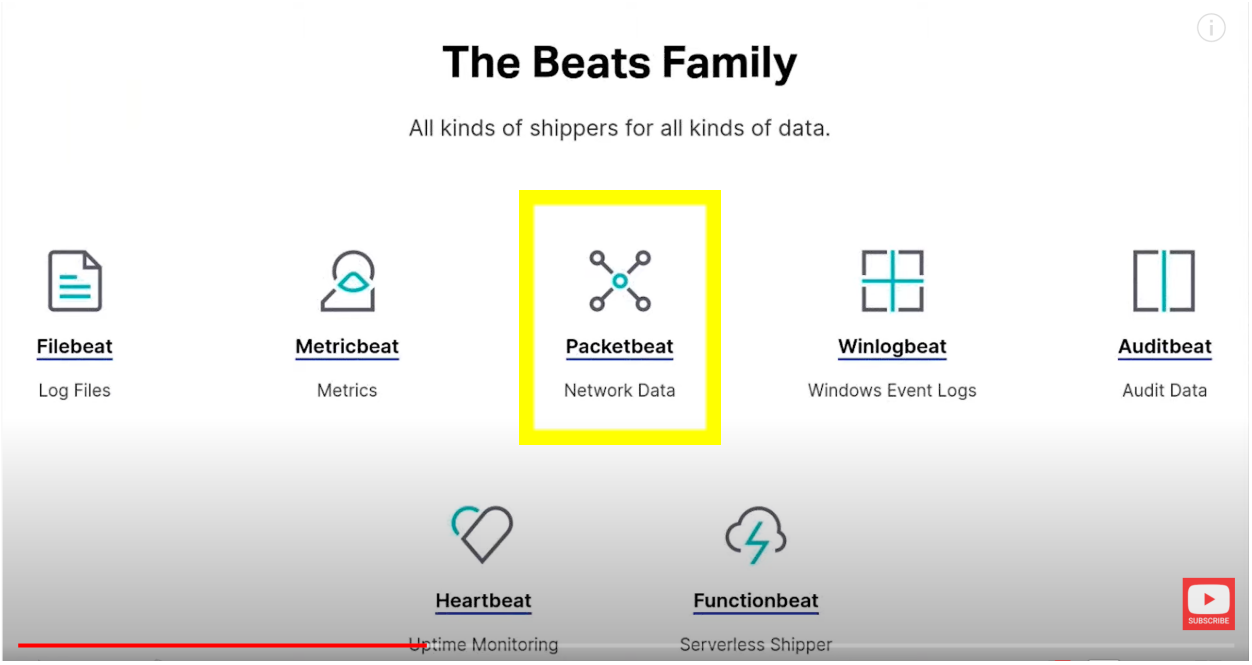
Beats

Beats are single-purpose, lightweight data shippers that we use to get data from different servers.

Beats can be installed on the servers as a lightweight agent to send system metrics, or process or file data to Logstash or Elasticsearch.

They gather data from the machine on which they are installed and then send that data to Logstash, which we use to parse or transform the data before sending it to Elasticsearch, or we can send the Beats data directly into Elasticsearch.

They are quite handy as it takes almost no time to install and configure



Excerpt from Appendix C.5 - Notes on Phoenixnap Elasticsearch Tutorial:
Phoenixnap ELK Stack Tutorial

What is the ELK Stack?

- Elasticsearch – The core component of ELK, which works as a searchable database for log files
- Logstash – A pipeline to retrieve data, which can be configured to retrieve data from many different sources and then to send to Elasticsearch
- Kibana – A visualization tool which uses a web browser interface to organize and display data.
- Beats – Smaller data collection applications which are specialized for individual tasks
    - Packetbeat is used to analyze network traffic

How Does Elastic Stack Work?

1. A computer or server creates log files
    a. All computers have log files that document events on the system
    b. Elastic Stack is designed to help manage scalable amounts of log data
2. The log files are collected by a Beats application (can skip to Logstash)
3. Logstash is configured to collect data from Beats applications
    a. Possible to collect and filter data from multiple systems
4. Logstash pipes data to Elasticsearch (scalable, searchable database)
5. Kibana provides GUI to configure and review data

Elasticsearch Overview

Two main jobs:

1. Storage and indexing of data
2. Search engine to retrieve data

Technical details:

- Robust programming language support for clients (Java, PHP, Ruby, C#, Python)
- Uses a REST API – Applications written for Elasticsearch have excellent compatibility with Web applications
- Responsive results – Users see data almost in real-time
- Distributed architecture – Elasticsearch can run and connect between many different servers. The Elastic Stack can scale easily as infrastructure grows.

- Inverted indexing – Elasticsearch indexes by keywords, much like the index in a book. This helps speed up queries to large data sets.
- Shards – If your data is too large for your server, Elasticsearch can break it up into subsets called Shards.
- Non-relational (NoSQL) – Elasticsearch uses a non relational database to break free from the constraints of structured/tabular data storage
- Apache Lucene – This is the base search engine that Elasticsearch is based on

Logstash

Logstash is a tool for gathering and sorting data from different sources used to sort, filter, and organize data. It collects a specific set of logs to import into Elasticsearch and contains default configurations with the alternative option of creating custom configurations

Technical features:

- Accepts a wide range of data formats and sources – This helps consolidate different data sets into one central location.
- Manipulates data in real-time – As data is read from sources, Logstash analyzes it and restructures it immediately.
- Flexible output – Logstash is built Elasticsearch, like many open-source projects, it can be reconfigured to export to other utilities
- Plugin support – A wide range of add-ons can be added to enhance Logstash's features.

Kibana

Kibana gives a GUI to generate and display data more intuitively

Technical features

- Dashboard interface – Configure graphs, data sources, and at-a-glance metrics.
- Configurable menus – Build data visualizations and menus to quickly navigate or explore data sets.
- Plug-ins – Adding plug-ins like Canvas allow you to add structured views and real-time monitoring to your graphical interface.

Beats

Beats runs on the system it monitors and collects and ships data to a destination (Logstash or Elasticsearch)

Resources Utilized: See Appendix C.6

Unity

Unity is a powerful game engine that uses C# and various assets (such as 3D elements rendered in programs like Blender) to create both 2D and 3D experiences. For this visualization, it will be utilized to visually represent the flow of network traffic between routers.

CentOS

For this visualization, we have used a CentOS 8 (red hat distribution of linux) Virtual Machine with the GUI install. For steps to set up Virtual Machine in the same configuration through Oracle Virtualbox, see Appendix C.8

Docker

Docker is installed on our CentOS 8 VM and will run the Elasticsearch configuration. To install Docker, and Elasticsearch with Docker, utilize the following tutorials:

- [47]

Zeek

Zeek (formerly Bro) is a tool used to monitor the flow of data over a network.

Unity Project Structure

The existing Unity project structure for ErgoWitness contains

VERSION CONTROL

Although it is relatively simple to connect a Unity project to Version control, it is necessary to implement a couple of changes to settings and the .gitignore file to prevent temporary files from being pushed

Unity Gitignore ignores temporary files

- Template found here.

Unity Settings to change

- Make sure meta files aren't pushed
- Change version control settings
- Refer to the video tutorial for more assistance (here)

CLASSES/SCRIPTS

Classes exist to do the following:

1. Move packets between nodes
2. Manage known devices, and add if not included
3. Manage devices and group by IP
4. Monitor and query server
5. Monitor data using Packetbeat
6. Filter results with Logstash
7. Query server using Elasticsearch

3D ASSETS

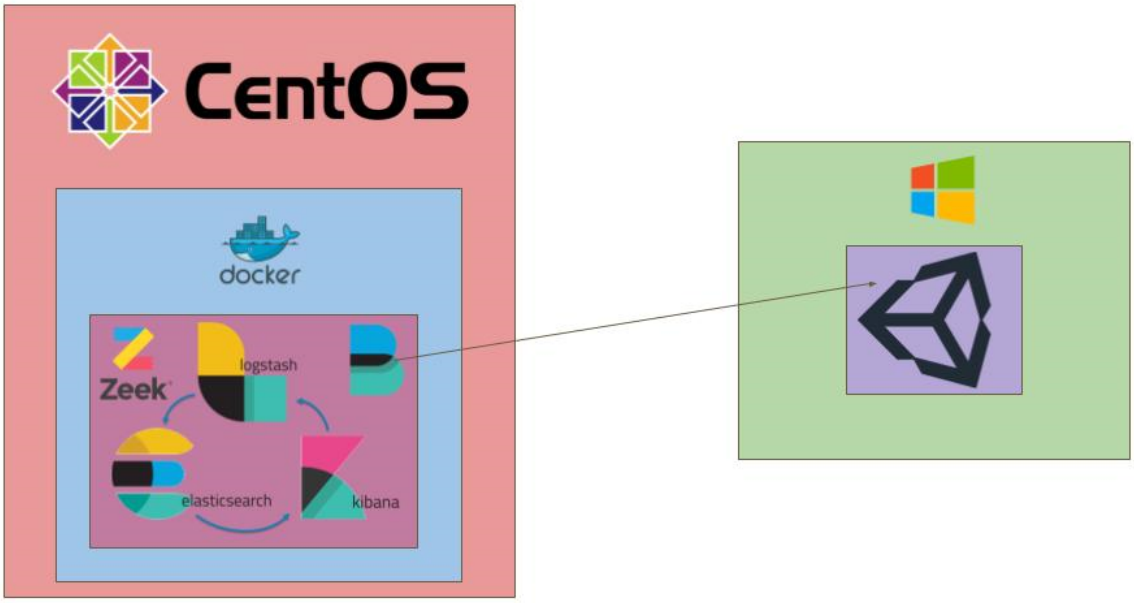
- Sphere (Network nodes)

- Basic box
- Cylinder
- paths

UNITY SCENES

1. Welcome/Menu
2. Main Visualization
3. Alternate UI/Test Space
4. Preloader for assets

How Does it Work?



The Unity game engine is run on a Windows 10 machine, and queries for network data. On a CentOS Virtual Machine, Elasticsearch is run inside of a Docker Container, and Zeek listens to network information, which is sent to the Unity C# code through curl requests. Although many of the scripts and versions have been updated, the tools used are largely the same.
The following images are the resources included with the ErgoWitness repository, including some scripts, which proved to be somewhat outdated in terms of versions, commands, and technology names. As a result, the initial VM setup was completed with the help of this additional tutorial: How to Install ELK Stack on CentOS 8 (phoenixnap.com)

Results

Network Simulation

For the virtual network, So far we have established our major components and all the configuration that is needed to set up a network simulator on a ubuntu physical or virtual computer. The procedures have been uploaded to the open-source repository for further testing. This procedure will allow you to create host pods within the cluster, and as well as deploy a randomly generated topology that can import VrNetlab router images.

Routing Protocol

The routing algorithm, routing configuration file, and hello protocol portions of the project were completed. The research, implementation, and design for each of these parts of the project have been documented and uploaded to the open-source repository. The routing algorithm and hello protocol have executable programs written for users to see how these parts of a routing protocol work.

Visualization

Thus far, we have CentOS 8 machine, which is successfully configured to run Elasticsearch inside of a Docker container. This base configuration will require adjustments in order to format and send network data in a useful format to our visualizer. While an initial version of the VM also ran Elasticsearch in its default configurations, this did not turn out to be an efficient way to organize the tools necessary. The new Virtual Machine is ready to configure with both Elasticsearch and Zeek, and hopefully the tutorials to change necessary log formatting will prove easier with this less haphazard install method. The Unity interface is set up on a Windows 10 PC, although further work is needed to format the commands from Elasticsearch through Packetbeat to be received by the visual interface.

Future Work

Network Simulation

To successfully complete the implementation for the simulation, we would need to finish creating a docker registry within the cluster to send images of our openWRT router from VrNetlab images. Once we can deploy pods containing openWRT routers, we can deploy a topology of the pods which can be accessed through the kubernetes.
To take it a step further we could also implement kind with kubernetes which essentially allows you to deploy the cluster inside a docker container rather than on the physical computer itself.

Once the network has successfully set up running openWRT routers within the network simulator we can import the routing protocol onto each pod and configure the routers to run the hello protocol.

Routing Protocol

The future work for the routing protocol portion of this project include finishing up the remaining components of a routing protocol:

- Hello Protocol Neighbor Recovery
- Flooding Protocol
- Metric Calculation
- Routing Table Creation

Once these components are completed, the routing protocol portion of the will be able to be connected to the virtual network project. This will be done by configuring the routers in the network to use the Hello Protocol program.

Another goal would be to adjust the Hello Protocol program to be able to retrieve its own ip address without needing it to be input by the user.

Network Visualization

The next steps to successfully combine the machines of this visualization are configuring the Virtual Machine’s Elasticsearch yml files to process, parse, and send the data to the Unity Visualizer. Once the files are configured correctly, then modifications can be made to the rate at which information is sent, as well as some UI tweaks.

Acknowledgements

We would like to thank :

- **Dr. Shereen Khoja** for her guidance during the planning and execution of this project
- **Dr. Chadd Williams** for providing the computer networking knowledge to inspire this project
- **Professor Doug Ryan** for always lending an ear!
- **Professor Chris Lane** for sharing his love of algorithms.
- **Ben Hoffman** for the use and instructions regarding his Ergo Witness project
- **Michael Kashin** - developer of k8s-topo and meshnet-CNI

References

Project Overview

[1] B. Hoffman, “BenjaFriend/ErgoWitness,” GitHub, Nov. 09, 2020. https://github.com/BenjaFriend/ErgoWitness (accessed Apr. 28, 2021).

[2] Cisco Networking Academy. 2014. Routing Protocols Companion Guide. Cisco Press.

[3] Cisco. Route Selection in Cisco Routers. (January 2008). Retrieved October 3, 2020 from https://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/8651-21.html

[4] Colorado State University. 2016. Project 3: Simulating a Link State Routing Protocol. Retrieved October 5th, 2020 from https://www.cs.colostate.edu/~cs457dl/yr2016sp/more_assignments/proj3.html

[5] Snorri Gylfason 1997. Link-State Routing. An Engineering Approach to Computer Networking. S. Keshav. Corporate and Professional Publishing Group. Retrieved October 5th, 2020 from http://www.cs.cornell.edu/skeshav/book/routing2/

[6] Dingwen Wang. 2013. Implementation of Link-State Routing Protocol (CS 542 Term Project, Fall 2013). Retrieved October 5th, 2020 from https://github.com/chuan6/link-state-routing

[7] Pengcheng Yin. 2012. OSPF_Router. Retrieved October 5th, 2020 from https://github.com/pcyin/OSPF_Router

[8] Robert Higgins, Jarad Cox, and Johnathan White. 2014. Network Simulation Application. Southern Illinois University. Retrieved October 6th, 2020 from https://www2.cs.siu.edu/~networksim/faq.html

Virtual Network

[9]“What is kubernetes?”, *Kubernetes*, 01-Feb-2021 [Online] available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[10]S. Simic, “Docker image vs container: The Major Differences”, *PhoenixNAP*, 31-Oct-2019. [Online]. Available: https://phoenixnap.com/kb/docker-image-vs-container

[11]S. biradar, “The Ultimate Docker Cheat Sheet”, *Collabnix*, [Online]. Available: https://dockerlabs.collabnix.com/docker/cheatsheet/

[12]“Docker overview”, *Docker,* . [Online]. Available: https://docs.docker.com/get-started/overview/

[13]K. Larsson, “vrnetlab - VR Network Lab”,*Github*, 01-Jan-2021,[Online]. Available: https://github.com/plajjan/vrnetlab

[14]“Vrnetlab: Emulate networks using KVM and Docker”, *Brianlinkletter*, 15-Mar-2019. [Online]. Available: https://www.brianlinkletter.com/2019/03/vrnetlab-emulate-networks-using-kvm-and-docker/

[15]“Kubernetes Components”, *Kubernetes*, 18-Mar-2021, [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/

[16]“Kubernetes should work with swap enabled”, *Github*, [Online] Available: https://github.com/kubernetes/kubernetes/issues/53533

[17]M. Kashin, “Meshnet CNI”, *Github*, [Online] Available: https://github.com/networkop/meshnet-cni

[18]“Koko”, Github, [Online] Available: https://github.com/redhat-nfvpe/koko

[19]M. Kashin, “k8s-topo”, *Github*, [Online] Available: https://github.com/networkop/k8s-topo

Routing

[20]“Routing,” *Wikipedia*, 10-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/Routing. [Accessed: 29-Apr-2021].

[21]Radiocrafts, “Why is Multicasting Becoming Essential for Mesh Networks?,” *Radiocrafts*, 26-Mar-2020. [Online]. Available: https://radiocrafts.com/why-is-multicasting-becoming-essential-for-mesh-networks/. [Accessed: 29-Apr-2021].

[22]“Cisco Networking Academy's Introduction to Routing Dynamically,” *Cisco Press*, 24-Mar-2014. [Online]. Available: https://www.ciscopress.com/articles/article.asp?p=2180210&seqNum=4. [Accessed: 29-Apr-2021].

[23]“Minimum Spanning Tree,” *Wikipedia*, 01-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/Minimum_spanning_tree#:~:text=A%20minimum%20spanning%20tree%20. [Accessed: 29-Apr-2021].

[24]G. Wu, *Minimum Spanning Tree*. 2020.

[25]G. Wu, *Maximum Spanning Tree*. 2020.

[26]“Kruskal's Algorithm,” *Programiz*. [Online]. Available: https://www.programiz.com/dsa/kruskal-algorithm. [Accessed: 29-Apr-2021].

[27]“Find if there is a path between two vertices in a directed graph,” *GeeksforGeeks*, 19-Jan-2021. [Online]. Available: https://www.geeksforgeeks.org/find-if-there-is-a-path-between-two-vertices-in-a-given-graph/. [Accessed: 29-Apr-2021].

[28]V. Joshi, *Depth-first search as compared to breadth-first search*. Medium, 2017.

[29]S. Sharma, “Print all the cycles in an undirected graph in C++,” *tutorialspoint*, 17-Jan-2020. [Online]. Available: https://www.tutorialspoint.com/print-all-the-cycles-in-an-undirected-graph-in-cplusplus. [Accessed: 29-Apr-2021].

[30]“Print all the cycles in an undirected graph,” *GeeksforGeeks*, 02-Nov-2020. [Online]. Available: https://www.geeksforgeeks.org/print-all-the-cycles-in-an-undirected-graph/. [Accessed: 29-Apr-2021].

[31]“Longest path problem,” *Wikipedia*, 21-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/Longest_path_problem. [Accessed: 29-Apr-2021].

[32]“7.1. The Hello Protocol,” *Connected: An Internet Encyclopedia*. [Online]. Available: https://www.freesoft.org/CIE/RFC/1583/28.htm. [Accessed: 29-Apr-2021].

[33]J. Moy, “OSPF Version 2,” *IETF*, Apr-1998. [Online]. Available: https://www.ietf.org/rfc/rfc2328.txt. [Accessed: 29-Apr-2021].

[34]R. Brown, “Welcome to the OpenWrt Project,” *OpenWrt Wiki*, 26-Apr-2021. [Online]. Available: https://openwrt.org/. [Accessed: 29-Apr-2021].

[35]“OpenWrt,” *Wikipedia*, 19-Apr-2021. [Online]. Available: https://en.wikipedia.org/wiki/OpenWrt. [Accessed: 29-Apr-2021].

[36]A. Bursi, “The build system,” *OpenWrt Wiki*, 22-Apr-2021. [Online]. Available: https://openwrt.org/docs/guide-developer/build-system/start. [Accessed: 29-Apr-2021].

Visualization

[37]“Visualization Research and Sources,” Google Docs. https://docs.google.com/document/d/1L-zmC0jOeYg2CDNAohGmwOZ3CIwOirBSypYIEfl9JKM/edit?usp=sharing (accessed Apr. 29, 2021).

[38]“Basic Concepts | Elasticsearch Guide [5.1] | Elastic,” www.elastic.co. https://www.elastic.co/guide/en/elasticsearch/reference/5.1/_basic_concepts.html (accessed Apr. 28, 2021).

[39]“Common Elastic Stack & Elasticsearch Architectures,” www.youtube.com. https://www.youtube.com/watch?v=Yc-G13lEbpc&t=263s (accessed Apr. 28, 2021).

[40]“Elasticsearch Tutorial | ELK Stack Tutorial | Intellipaat,” www.youtube.com. https://www.youtube.com/watch?v=cC4GGJ0JsSE&t=2983s (accessed Apr. 28, 2021).

[41]“Elasticsearch Tutorial & Getting Started (course preview),” www.youtube.com. https://www.youtube.com/watch?v=ksTTlXNLick (accessed Apr. 28, 2021).

[42]“ELK Stack Tutorial: Getting Started With the Elastic Stack,” Knowledge Base by phoenixNAP, Aug. 04, 2020. https://phoenixnap.com/kb/elk-stack-tutorial (accessed Apr. 28, 2021).

[43]“How to Install ELK Stack on CentOS 8,” Knowledge Base by phoenixNAP, May 06, 2020. https://phoenixnap.com/kb/install-elk-stack-centos-8#htoc-add-the-elasticsearch-rpm-repository (accessed Apr. 28, 2021).

[44]“Introduction to Elasticsearch,” www.youtube.com. https://www.youtube.com/watch?v=yZJfsUOHJjg.

[45]“Overview of the Elastic Stack (formerly ELK stack),” www.youtube.com. https://www.youtube.com/watch?v=Hqn5p67uev4.

[46]“What is ELK Stack | ELK Tutorial For Beginners | Elasticsearch Kibana | ELK Training | Intellipaat,” www.youtube.com. https://www.youtube.com/watch?v=LDAIzpUJItg (accessed Apr. 28, 2021).

[47]“Install Elasticsearch with Docker | Elasticsearch Guide [7.12] | Elastic,” www.elastic.co. https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html (accessed May 07, 2021).

Appendices

This section will include all of the documentation, notes, and research we compiled while completing this project. Virtual Network appendices in section A, Routing Protocol appendices in section B, Network Visualizer appendices in section C

A - Virtual Network

Scenic-Routing/Network Simulation at main · crumbj/Scenic-Routing (github.com)

B - Routing Protocol

https://github.com/crumbj/Scenic-Routing
https://github.com/crumbj/Scenic-Routing/tree/main/Routing%20Protocol

C - Network Visualizer

C.1 Visualization Platform Research

Unity

Visualizing 3D Network Topologies Using Unity (9/26/20)

- Some source code using C Sharp for Network Data Visualization

Beginning Data Visualization in Unity: Scatterplot Creation – Methodology blog (9/26/20)

- Basic Unity setup and data visualization with scatter plot

Graphics Examples (9/26/20)

- No source code

Other Technologies:

Harp.gl (10/11/20)

https://www.harp.gl/

https://www.harp.gl/docs/master/examples/#getting-started_look-at.html

- Web-based, open source visualization

- maps/globe/direction

https://ipfabric.io/blog/routing-protocol-visualization/ (10/11/20)

- Blog post about a network visualization tool
- No source code or demo videos
- VIsual organization could be a helpful tool.

IMPORTANT

- Minimal source code is available. The only one I could find is written in C-sharp with Unity.
- Unity workspace looks easy to set up, but time-intensive with VS install.
- Still need to find out if Python works with Unity.
    - Example with C-sharp
- Need to determine how data is stored (new structure, built in structure, text?)
    - C-sharp example has example of node structure

Network Visualization Example Videos:

Neural Network Visualization using Unity and C-Sharp (10/2/20)

Tutorials and Source Code:

R with HTML/JavaScript animation (10/2/20)

- This actually includes a lot of the source code and libraries needed to create a network simulation and graphing/plotting data in R.
- The examples are less interactive than I would like and don't seem to use any kind of real-time data.

Hello World Tutorial Unity (10/11/20)

- How to create a cs script as an asset, and run in a Unity project

Python with Unity (10/12/20)

Scripting Tutorials (10/12/20

Documentation:

Unity Documentation (10/11/20)

C# Documentation (10/11/20)

Python in Unity (10/11/20)

Unity Learn (10/11/20)

Steps to Download:

1. Apply for Unity student
2. Download Unity hub and editor (for now, using VS code)
3. Activate license using emailed key
4. Create new project and follow Hello World Tutorial
5. Python installation

Current Decision:

- Work with Unity platform for graphics
- Languages: Python (if integration works) and C#
- Figure out if Unity projects can be implemented on a website.
- Next steps:
    - Set up a sample project and start looking at assets
    - Script with node class/data structure

C.2 Github Setup for Unity
Unity Gitignore ignores temporary files

- Template found here.

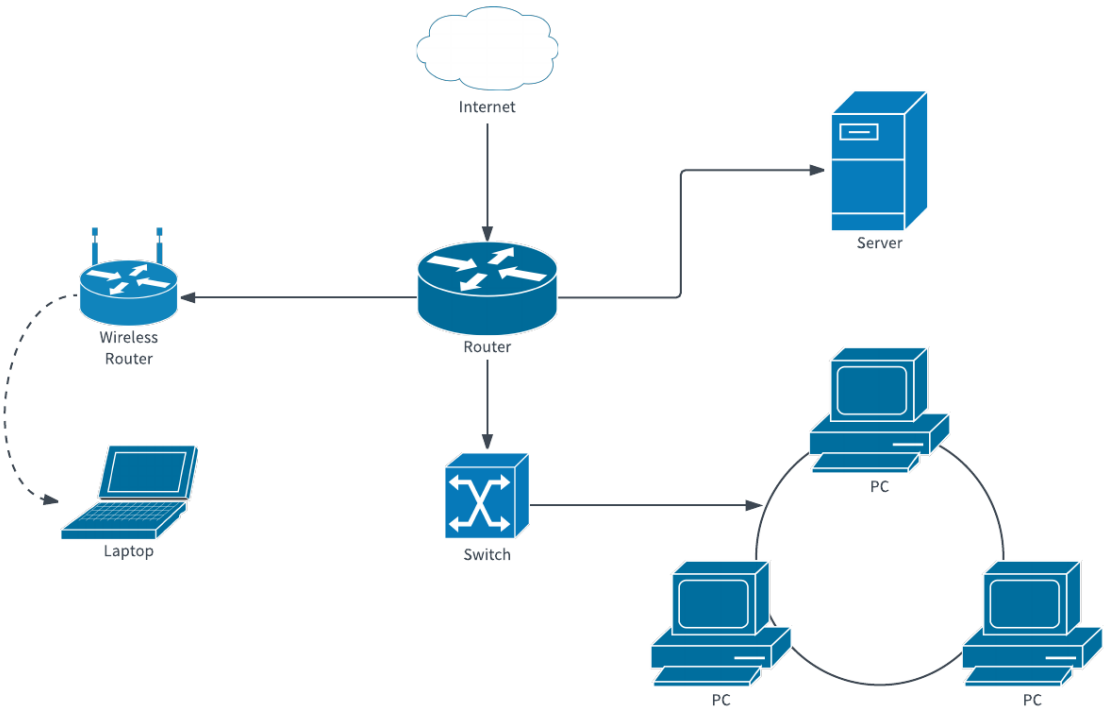
Unity Settings to change

- Make sure meta files aren't pushed
- Change version control settings
- Video here

C.3 Network Information Flow

Jonathan Strickland "How IP Convergence Works" 8 March 2010. HowStuffWorks.com. <https://computer.howstuffworks.com/ip-convergence.htm> 22 October 2020

- Information travels in packets (max byte size).
- Packets do not have to take the same path to the destination
- Once all packets have arrived at the destination, they are reassembled (puzzle analogy)



https://github.com/BenjaFriend/ErgoWitness/wiki

C.4 Unity Project Structure

Unity Project Structure

Assets needed to complete project

Source: ErgoWitness

VERSION CONTROL

Unity Gitignore ignores temporary files

- Template found here.

Unity Settings to change

- Make sure meta files aren't pushed
- Change version control settings
- Video here

CLASSES/SCRIPTS

1. Move packets between nodes
2. Device Manager
    a. Manage known devices, add if not included
3. Manage devices and group by IP?
4. Monitor and query server
5. Data Monitoring (example used PacketBeat)
6. Logstash filter for Elasticsearch
7. ElasticSearch/querying server
8. Snort?
    a. MORE RESEARCH NEEDED

3D ASSETS

- Sphere (Network nodes)
- Basic box
- Cylinder
- paths

UNITY SCENES

1. Welcome/Menu
2. Main Visualization
3. Alternate UI/Test Space
4. Preloader for assets

TECHNOLOGIES TO SET UP

Research these and others like them to determine best to use

1. Logstash
2. ElasticSearch/Elk Stack
3. IP addresses for our network

Notes

NOTES ON SCRIPTS

All have the following includes:

```
Using
System;

                using System.Collections;

                using
                System.Collections.Generic;

                using UnityEngine;
```

Computer:

- Computer.cs
- ComputerUI.cs
- Computer_AnimationController.cs

Gravity Testing:

NODE:

- Node.cs
    - Mouse grab and drag
    - Connecting joint from source to destination
    -

IP Groups:

- IPGroup.cs
  -
- IPGroupManager.cs
  - Arrays for color groups
  - Static reference to this manager
  - Game object: prefab for IP group
  - Floats: minimum distance apart, size, increase (radius) amount per group
  - Int: last color used
    - This code has option to randomly assign color
  - Temp references
    - GameObject
    - IP group
  - Int: attempt count
  - Functions
    - `private void Awake()`
      - Checks that only one IPGroupManager is awake
    - `void Start()`
      - Sets up static reference and initializes groups
    - `private int[] ReadInIps(string fileLocation)`
      - Read from file list of IP addresses
      - fileLocation: location to read from
    - `public void SetIpsViaOptions(string[] ipAddress, int groupNum)`
    - `public void RemoveGroup(int groupToRemove)`
    - `public void CheckGroups(int ipToCheck)`
      - Checks if IP fits in existing group. If not, creates a new group and adds IP address
    - `private void MakeNewGroup(int ipToCheck, int groupIPFirstThree)`
      - Creates new IP group
    - `private void SetGroupPosition(GameObject moveMe)`
      - Checks for collisions with other groups, sets group position
    - `private void SetGroupColor(IPGroup groupToColor)`
    - `public int GetFirstThreeIpInt(int ipToCheck)`
      - Returns first three numbers of ip address
    - `private string IpIntToString(int ipAddrInt)`
    - `private int IpToInt(string ipAddr)`
    - `public IEnumerator HideAlertType(int alertType)`
      - Calculates alerts for each IP

Netflow Data:

- ConnectionController.cs

- DestroyAfterTime.cs
- MoveFromSourceToTarget.cs
- NetflowObject.cs
- NetflowPauseController.cs

ELK STACK TUTORIAL LINKS

https://phoenixnap.com/kb/elk-stack-tutorial

https://logz.io/learn/complete-guide-elk-stack/

ACTIVITY

- *Github repository and empty project created with initial commit (10/22/20)*

1.5 Structure notes on ErgoWitness

https://drive.google.com/file/d/18s-pVgbKNhWjTlxdYvfcrNR4_YQldKzo/view?usp=sharing

A.6 CentOS

1. Download mirror to SSD. http://isoredirect.centos.org/centos/8/isos/x86_64/
2. Follow tutorial steps until reboot: https://gigabytekingdom.com/install-centos-on-virtualbox/
3. Before reboot, delete .iso file from storage settings of machine: https://www.reddit.com/r/virtualbox/comments/dlzw7u/centos_8_keeps_reinstalling_on_reboot_after/

Other resources looked at:
https://onlinecomputertips.com/support-categories/software/662-virtualbox-boot-iso\
https://www.onlinecomputertips.com/support-categories/software/505-create-vm-virtualbox
https://laptop.ninja/how-to-setup-centos-in-virtualbox/#:~:text=%20Setup%20CentOS%20in%20VirtualBox%20%201%20Step,Virtual%20Machine.%20Click%20on%20your%20new...%20More%20

C.5 Elasticsearch and ELK Stack Notes
What is ELK Stack | ELK Tutorial For Beginners | Elasticsearch Kibana | ELK Training | Intellipaat - YouTube

What is ELK?

Elastic Stack is a group of open source products from Elastic (company) designed to help users take data from any type of source and in any format and search, analyze, and visualize that data in real time.

The heart of Elastic Stack is Elasticsearch, which drives everything.

Elastic Stack presents a steeper learning curve than some comparable products (you need to set up everything), as well as more set up, owing in part to its open source nature. In return for the extra work, however, the sysadmin is rewarded with a deeper understanding of the software's underlying structure.

Elastic was founded in Amsterdam in 2012 to support the development of Elasticsearch and related commercial products and services.

Elasticsearch, Logstash, Kibana are the three main components

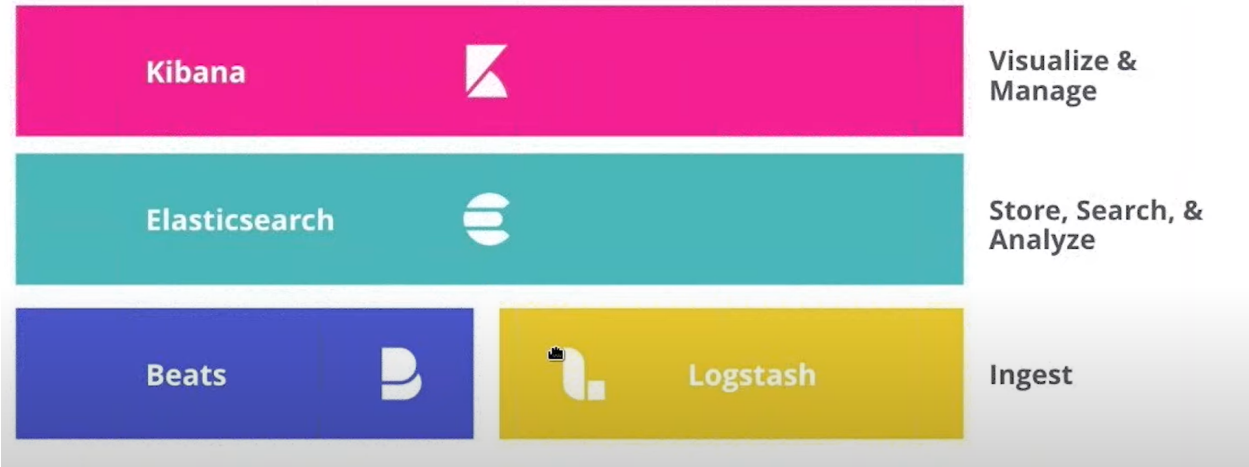
Components of Elastic Stack

Elasticsearch: We can store data in Elasticsearch

Logstash: A data pipeline that we can use to read data from various sources, and can write it to various sources. It also provides a feature to filter the input data before sending it to output

Kibana: A graphical user interface that we can use to do a lot of things

Beats: Lightweight data shippers that sit on different servers and send data to Elasticsearch directly or via Logstash



ELK Stack to Elastic Stack

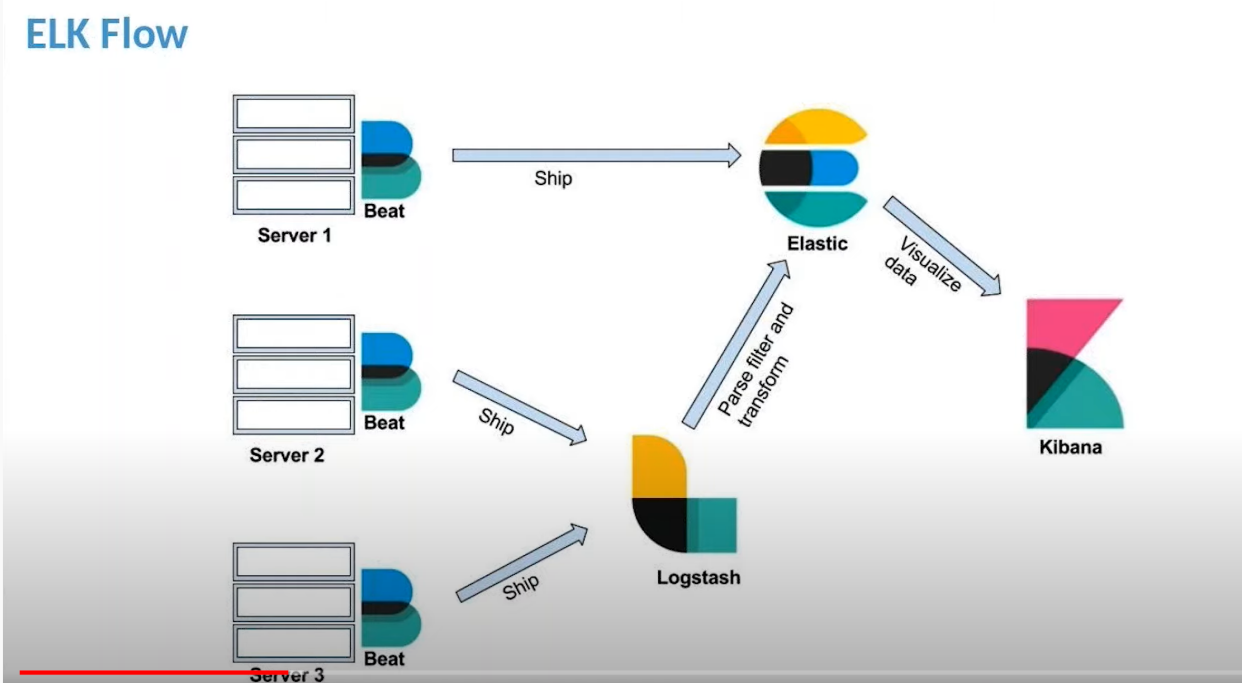
It all started with Elasticsearch:

The open source, distributed, RESTful, JSON-based search engine. Easy to use, scalable, and flexible

Then it grew with Logstash and Kibana

Later as the community expanded and the use cases increased. They now added Beats on ELK, so this brought one additional tool.

The Elastic Stack is the ELK Stack, but with more flexibility to do great things.



Elasticsearch

Elasticsearch is a full-text search engine that's primarily used for searching. It can also be used as a NoSQL database and analytics engine.

Elasticsearch is basically schema-less and works in near-real time.

It has a RESTful interface, which helps us to interact with it easily from multiple interfaces.

Elasticsearch supports different ways of importing various types of structured or unstructured data; it handles all types of data because of its schema-less behavior and it's quite easy to scale.

Kibana

In Elastic Stack, Kibana is mainly used to provide the graphical user interface, which we use to do multiple things.

When Kibana was first released, we just used it to create charts and histograms, but with each update, Kibana evolves and now we have lots of killer features that make Kibana stand out from the crowd.

There are many features in Kibana, but when we talk about the key features,

1. Discover your data by exploring it
2. Analyze your data by applying different metrics
3. Visualize your data by creating different types of charts
4. Apply machine learning on your data to get data anomaly and future trends
5. Monitor your application using APM (Application Performance Monitoring)
6. Manage users and roles
7. A console to run Elasticsearch expressions
8. Play with time-series data using Timeline
9. Monitor your Elastic Stack using Monitoring

Logstash

Logstash is a data pipeline that can take data input from various sources, filter it, and out put it to various sources; these sources can be files, Kafka, or databases.

Logstash is a very important tool in Elastic Stack as it's primarily used to pull data from various sources and push it to Elasticsearch; from there, Kibana can use that data for analysis or visualization.

We can take any type of data using Logstash, such as structured or unstructured date, which comes from various sources, such as the internet.

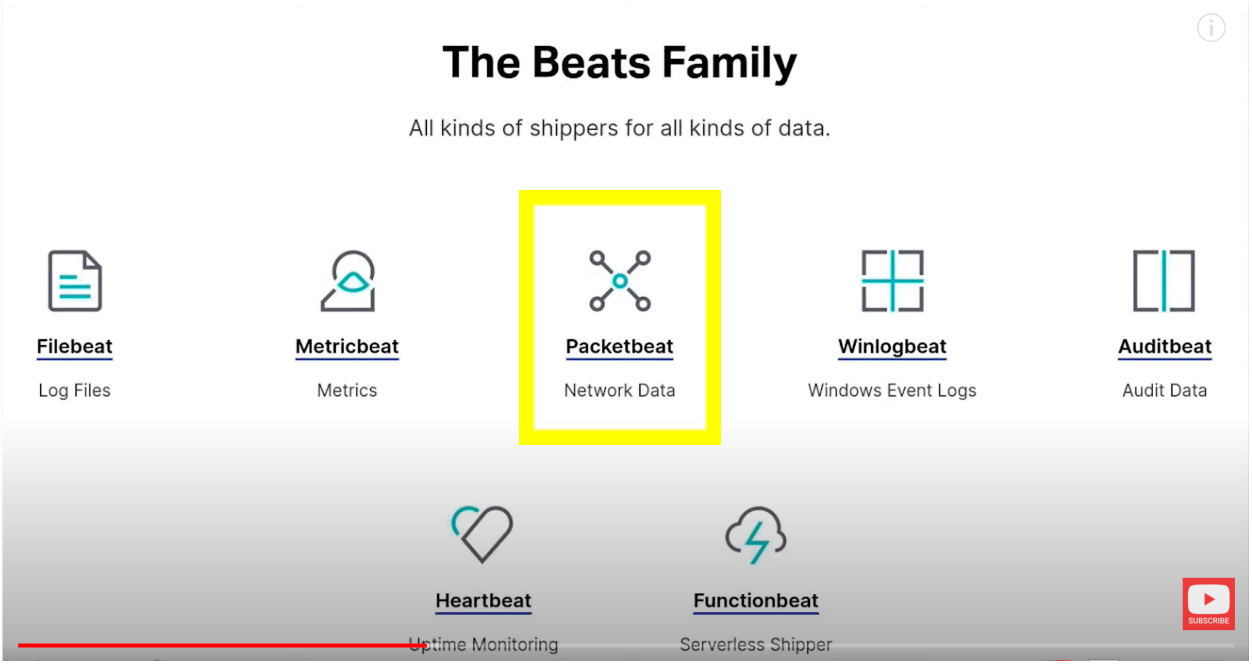
Beats

Beats are single-purpose, lightweight data shippers that we use to get data from different servers.

Beats can be installed on the servers as a lightweight agent to send system metrics, or process or file data to Logstash or Elasticsearch.

They gather data from the machine on which they are installed and then send that data to Logstash, which we use to parse or transform the data before sending it to Elasticsearch, or we can send the Beats data directly into Elasticsearch.

They are quite handy as it takes almost no time to install and configure

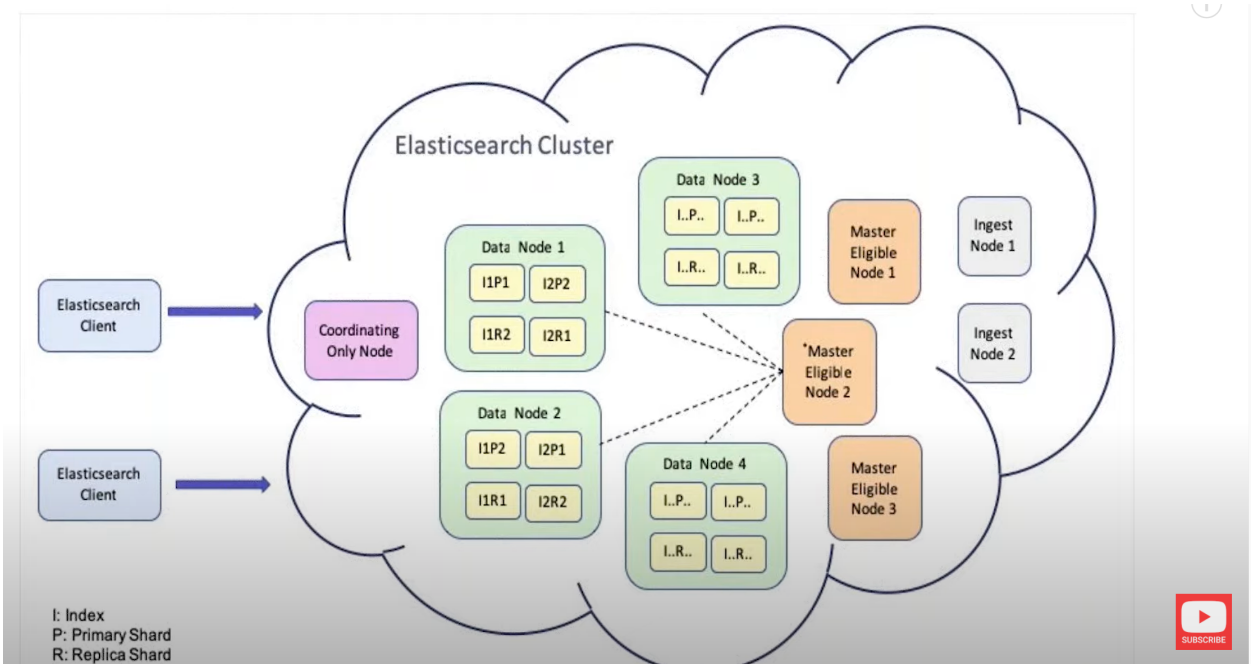


Elasticsearch architecture

Elasticsearch is a real-time distributed search and analytics engine with high availability. It is used for full-text search, structured search, analytics, or all three in combination. It is built on top of the Apache Lucene library.

It is a schema-free, document-oriented data store. However, unless you fully understand your use case, the general recommendation is not to use it as the primary data store. One of the advantages is that the RESTful API uses JSON over HTTP, which allows you to integrate, manage, and query index data in a variety of ways.

An **Elasticsearch cluster** is a group of one or more Elasticsearch nodes that are connected together. Let’s first outline how it is laid out, as shown in the following diagram:



Although each node has its own purpose and responsibility, each node can forward client requests (coordination) to the appropriate nodes. The following are the nodes used in an Elasticsearch cluster:

- **Master-eligible node:** The master node's tasks are primarily used for lightweight cluster-wide operations, including creating or deleting an index, tracking the cluster nodes, and determining the location of the allocated shards. By default, the master-eligible role is enabled. A master-eligible node can be elected to become the master node (the node with the asterisk) by the master-election process. You can disable this type of role for a node by setting node.master to false in the elasticsearch.yml file.
    - **2-**3
- **Data node:** A data node contains data that contains indexed documents. It handles related operations such as CRUD, search, and aggregation. By default, the data node role is enabled, and you can disable such a role for a node by setting the node.data to false in the elasticsearch.yml file.
    - many
- **Ingest node:** using an ingest node is a way to process a document in pipeline mode before indexing the document. By default, the ingest node role is enabled—you can disable such a role for a node by setting node.ingest to false in the elasticsearch.yml file.
- **Coordinating-only node:** If all three roles (master eligible, data, and ingest) are disabled, the node will only act as a coordination node that performs routing

requests, handling the search reduction phase, and distributing works via bulk indexing.

1. Fields

   Fields are the smallest individual unit of data in Elasticsearch. These are customizable and could include, for example: title, author, date, summary, team, score, etc.

   Each field has a defined datatype and contains a single piece of data. Those datatypes include the core datatypes (strings, numbers, dates, booleans), complex datatypes (object and nested), geo datatypes (get_point and geo_shape), and specialized datatypes (token count, join, rank feature, dense vector, flattened, etc.)

   There are different kinds of fields and ways to manage them.

   Multi-fields

   These fields can (and should) be indexed *in more than one way* to produce more search results. This option is available easily through the “multi-fields” option, which as you might have guessed allows for fields to be indexed in multiple ways. For example, something might be indexed either as free *text* or a specific *keyword*.

   Meta-fields

   > Meta-fields deal with a document’s metadata. This variety of fields will be further elaborated on under the *Mapping* heading.

2. Documents

   Documents are JSON objects that are stored within an Elasticsearch index and are considered the base unit of storage. In the world of relational databases, documents can be compared to a row in a table.

   For example, let’s assume that you are running an e-commerce application. You could have one document per product or one document per order. There is no limit to how many documents you can store in a particular index.

Elasticsearch	RDBMS
Index	Database
Shard	Shard
Mapping	Table
Field	Field
JSON Object	Tuple

Data in documents is defined with fields comprised of keys and values. A key is the name of the field, and a value can be an item of many different types such as a string, a number, a Boolean expression, another object, or an array of values.

Documents also contain reserved fields that constitute the document metadata such as:

- _index – the index where the document resides
- _type – the type that the document represents
- _id – the unique identifier for the document

An example of a document:

```
{
   "_id": 3,
   "_type": ["your index type"],
   "_index": ["your index name"],
   "_source":{
   "age": 28,
   "name": ["daniel"],
   "year":1989,
}
}
```

3. Mapping

As far as mapping goes, bear in mind that since Elasticsearch 7.0, *index type* has been deprecated. The following is still relevant to legacy versions of Elasticsearch. *Data type* is still active.

Like a schema in the world of relational databases, mapping defines the different *types* that reside within an index (although for 6.0 until its deprecation in 7.0, only one type can exist within an index). It defines the *fields* for documents of a specific type—the data type(such as keyword and integer) and how the fields should be indexed and stored in Elasticsearch. This includes meta-fields, such as _index, _type, and _id.

```
# Example
curl -XPUT localhost:9200/example -d '{
  "mappings": {
    "mytype": {
      "properties": {
        "name": {
          "type": "string"
        },
        "age": {
          "type": "long"
        }
      }
```

4. Mapping Types

   Not to be confused with *datatypes*, mapping *types* are now a legacy aspect of Elasticsearch, related to all previous versions released prior to Elasticsearch 7.0.0. However, they remain critical when dealing with those previous iterations. Each mapping type could have had its own field. This led to confusion because fields with the same name even in completely separate mapping types would have to be simultaneously defines as the same *datatype* (even though the *datatypes* would not necessarily match).

5. Indices

   Indices, the largest unity of data in Elasticsearch, are logical partitions of documents and can be compared to a database in the world of relational databases.

   Continuing our e-commerce app example, you could have one index containing all of the data related to the products and another with all of the data related to the customers.

   You can have as many indices defined in Elasticsearch as you want. These in turn will hold documents that are unique to each index.

Indices are identified by lowercase names that refer to actions that are *performed actions* (such as searching and deleting) ***on*** the documents that are inside each index.

For a list of best practices in handling indices, check out the blog Managing an Elasticsearch Index.

Another key element to getting how Elasticsearch's indices work is to get a handle on shards.

6. Shards

   Put simply, shards are a single Lucene index. They are the building blocks of Elasticsearch and what facilitate its scalability.

   Index size is a common cause of Elasticsearch crashes. Since there is no limit to how many documents you can store on each index, an index may take up and amount of disk space that exceeds the limits of the hosting server. As soon as an index approaches this limit, indexing will begin to fail.

   One way to counter this problem is to spit up indices horizontally into pieces called *shards*. This allows you to distribute operations *across shards **and** nodes* to improve performance.

   When you create an index, you can define how many shards you want. Each shard is an independent Lucene index that can be hosted anywhere in your cluster.

```
# Example
curl -XPUT localhost:9200/example -d '{
  "settings" : {
    "index" : {
      "number_of_shards" : 2,
      "number_of_replicas" : 1
    }
  }
}'
```

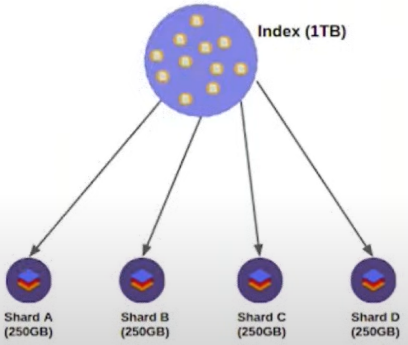
7. Replicas

   Replicas, as the name implies, are Elasticsearch fail-safe mechanisms and are basically copies of your index's shards. This is a useful backup system for a rainy day—or, in other words, when a node crashes. Replicas also serve read requests, so adding replicas can help to increase search performance.

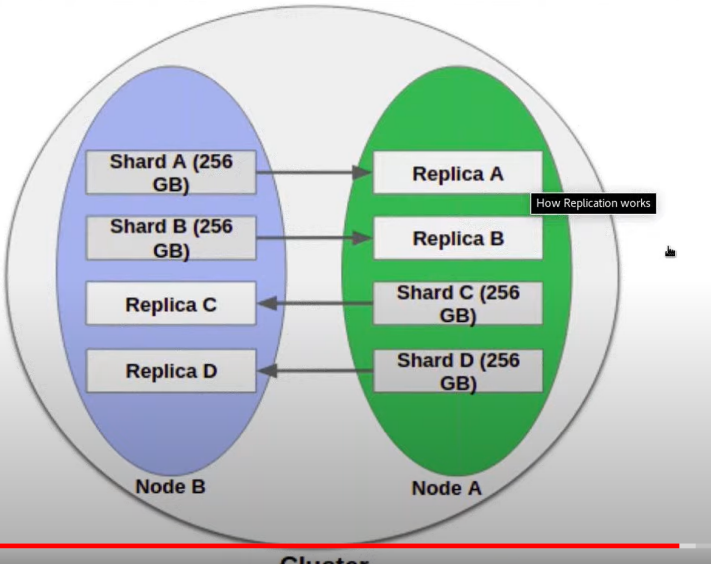
   To ensure high availability, replicas are not placed on the same node as the original shards (called the "primary" shard) from which they were replicated. For example, Shard 1 might share Node A with Replica Shard 2, while Node B hosts

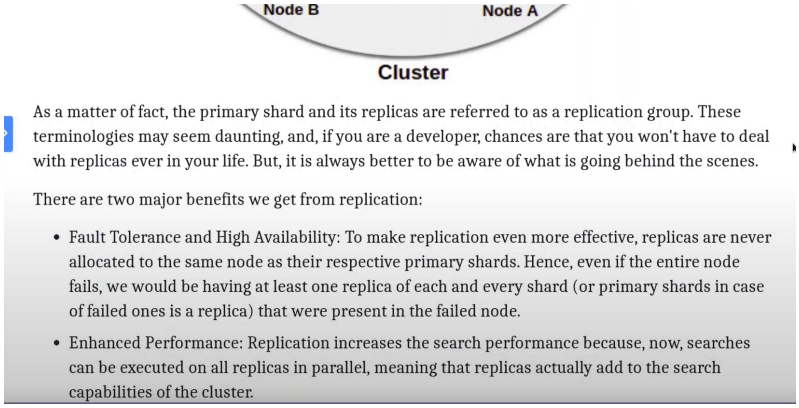
Shard 2 and Replica Shard 1. In a true use case, there will be several more shards.

In cases like this, where the size of an index exceeds the hardware limits of a single node, sharding comes to the rescue. Sharding solves this problem by dividing the indices into smaller pieces and these pieces are called shards.



As can be seen, a shard contains a subset of the index's data. When an index is sharded, a given document within that index will only be stored within one of the shards. The amazing thing about shards is that they can be hosted on any node within the cluster. Now, coming to our example, we can





Phoenixnap ELK Stack Tutorial

What is the ELK Stack?

- Elasticsearch – The core component of ELK, which works as a searchable database for log files
- Logstash – A pipeline to retrieve data, which can be configured to retrieve data from many different sources and then to send to Elasticsearch
- Kibana – A visualization tool which uses a web browser interface to organize and display data.
- Beats – Smaller data collection applications which are specialized for individual tasks
    - Packetbeat is used to analyze network traffic

How Does Elastic Stack Work?

1. A computer or server creates log files
    a. All computers have log files that document events on the system
    b. Elastic Stack is designed to help manage scalable amounts of log data
2. The log files are collected by a Beats application (can skip to Logstash)
3. Logstash is configured to collect data from Beats applications
    a. Possible to collect and filter data from multiple systems
4. Logstash pipes data to Elasticsearch (scalable, searchable database)
5. Kibana provides GUI to configure and review data

Elasticsearch Overview

Two main jobs:

1. Storage and indexing of data

2. Search engine to retrieve data

Technical details:

- Robust programming language support for clients (Java, PHP, Ruby, C#, Python)
- Uses a REST API – Applications written for Elasticsearch have excellent compatibility with Web applications
- Responsive results – Users see data almost in real-time
- Distributed architecture – Elasticsearch can run and connect between many different servers. The Elastic Stack can scale easily as infrastructure grows.
- Inverted indexing – Elasticsearch indexes by keywords, much like the index in a book. This helps speed up queries to large data sets.
- Shards – If your data is too large for your server, Elasticsearch can break it up into subsets called Shards.
- Non-relational (NoSQL) – Elasticsearch uses a non relational database to break free from the constraints of structured/tabular data storage
- Apache Lucene – This is the base search engine that Elasticsearch is based on

Logstash

Logstash is a tool for gathering and sorting data from different sources used to sort, filter, and organize data. It collects a specific set of logs to import into Elasticsearch and contains default configurations with the alternative option of creating custom configurations

Technical features:

- Accepts a wide range of data formats and sources – This helps consolidate different data sets into one central location.
- Manipulates data in real-time – As data is read from sources, Logstash analyzes it and restructures it immediately.
- Flexible output – Logstash is built Elasticsearch, like many open-source projects, it can be reconfigured to export to other utilities
- Plugin support – A wide range of add-ons can be added to enhance Logstash's features.

Kibana

Kibana gives a GUI to generate and display data more intuitively

Technical features

- Dashboard interface – Configure graphs, data sources, and at-a-glance metrics.
- Configurable menus – Build data visualizations and menus to quickly navigate or explore data sets.
- Plug-ins – Adding plug-ins like Canvas allow you to add structured views and real-time monitoring to your graphical interface.

Beats

Beats runs on the system it monitors and collects and ships data to a destination (Logstash or Elasticsearch)

C.6 ELK Stack Tutorial List

Coding Explained Videos:

- Elasticsearch Tutorial & Getting Started (course preview)
- Introduction to Elasticsearch
- Overview of the Elastic Stack (formerly ELK Stack)
- Common Elastic Stack & Elasticsearch Architectures

Phoenixnap Tutorials:

- How to Install ELK Stack on CentOS 8 (phoenixnap.com)
- ELK Stack Tutorial: Getting Started With the Elastic Stack (phoenixnap.com)

Intellipaat Tutorials:

- https://www.youtube.com/watch?v=cC4GGJ0JsSE&t=2983s
- https://www.youtube.com/watch?v=LDAIzpUJItg

Elasticsearch Tutorial & Getting Started (course preview) - YouTube
Basic Concepts | Elasticsearch Reference [5.1] | Elastic

C.7 Network Visualization Flow Chart Sketch

https://drive.google.com/file/d/18s-pVgbKNhWjTlxdYvfcrNR4_YQldKzo/view?usp=sharing

C.8 Install CentOS and Docker

1. Download mirror to SSD. http://isoredirect.centos.org/centos/8/isos/x86_64/
2. Follow tutorial steps until reboot, using GUI install:https://gigabytekingdom.com/install-centos-on-virtualbox/
3. Before reboot, delete .iso file from storage settings of machine: https://www.reddit.com/r/virtualbox/comments/dlzw7u/centos_8_keeps_reinstalling_on_reboot_after/

Other resources looked at:
https://onlinecomputertips.com/support-categories/software/662-virtualbox-boot-iso\
https://www.onlinecomputertips.com/support-categories/software/505-create-vm-virtualbox
https://laptop.ninja/how-to-setup-centos-in-virtualbox/#:~:text=%20Setup%20CentOS%20in%20VirtualBox%20%201%20Step,Virtual%20Machine.%20Click%20on%20your%20new...%20More%20