ch #5 Networking [part one]

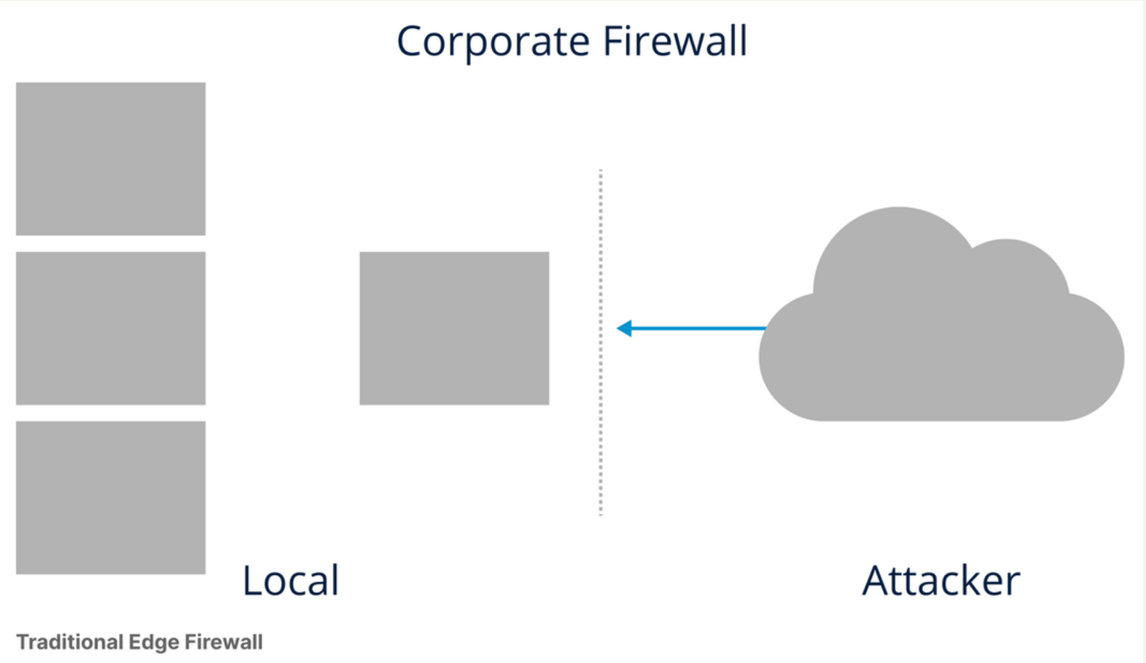
LFS 260

Networking Essentials

Every system needs protection

In too many production environments the issue of network security is pushed to the edge of the network. Instead of admins and devops expecting every node to be attacked by any of the other systems in the environment, they assume the opposite.

This assumption presumes those responsible are up to the task, and never make mistakes.



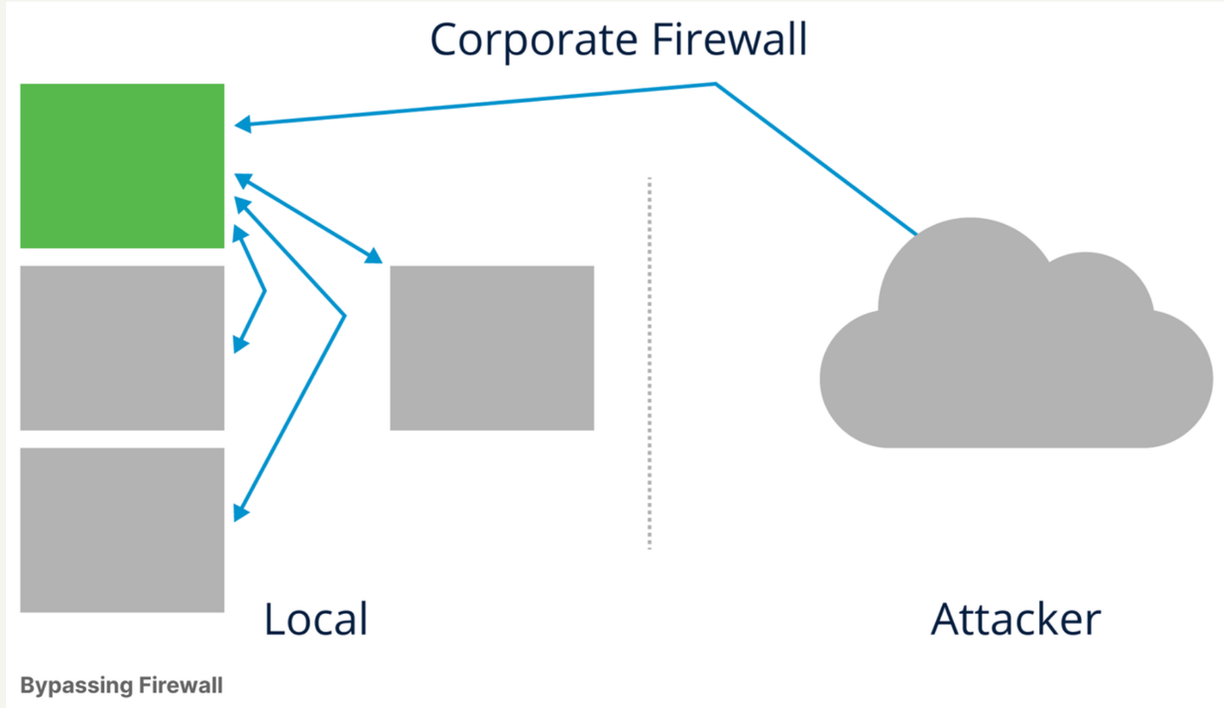
Traditional Edge Firewall

Networking Essentials

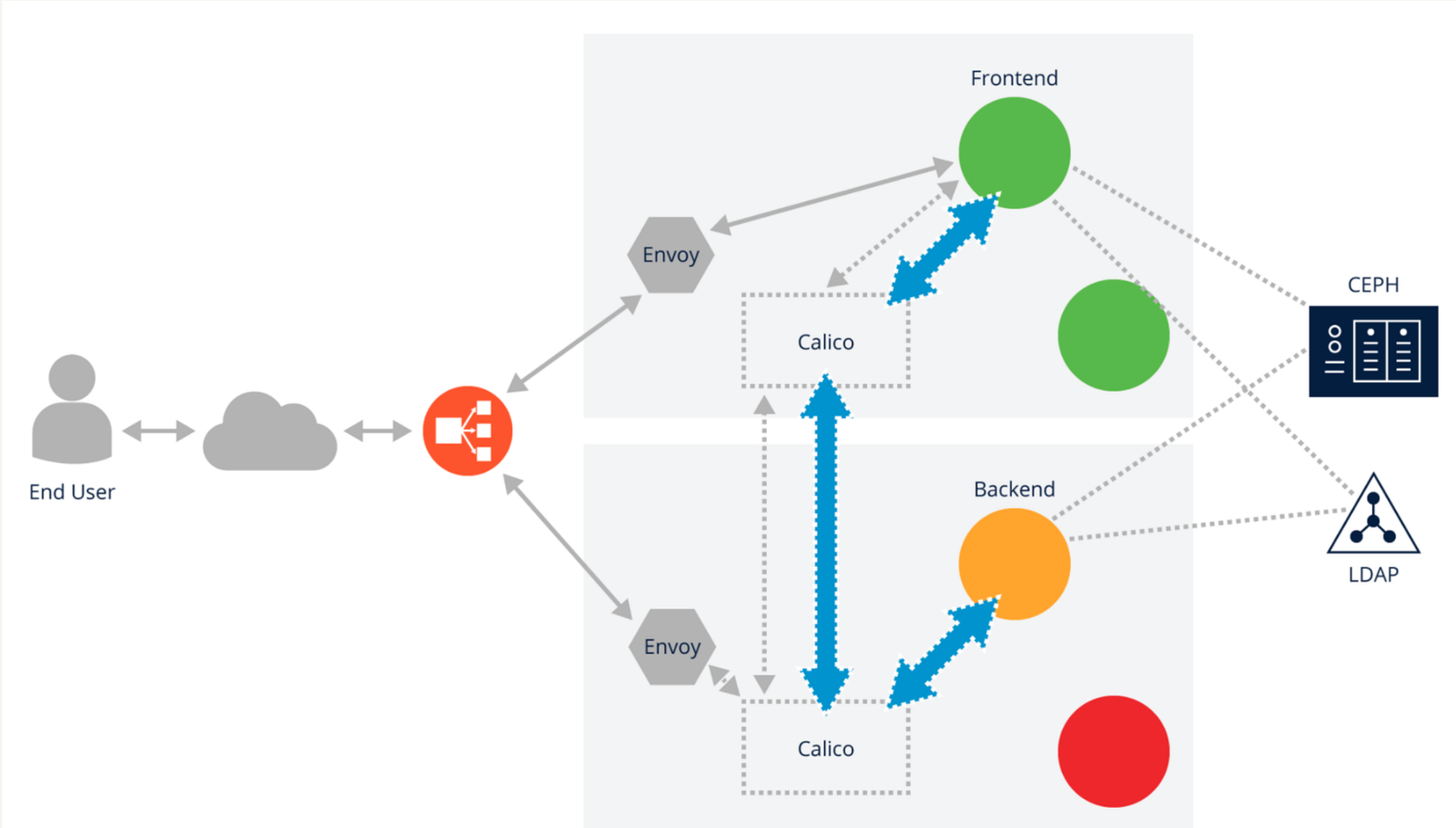
Unfettered Access

Should anyone find a way to gain access to any node, they could snoop network traffic or attack all of the nodes in the environment. Without network monitoring and logging, this attack could go on without anyone knowing where the attack is coming from or who is gathering their network traffic.

We need to protect the nodes within and at the edge of the network, to avoid an IT Maginot Line, with similar outcome.



Kubernetes Network



General View of Networking

Kubernetes Network

The end user on the left would make a request across the Internet often to an exterior load balancer such as HAProxy and/or F5. The load balancer would then pass along traffic to one of the worker nodes.

Which node would depend on its configuration and the needs of the applications the end user is trying to access.

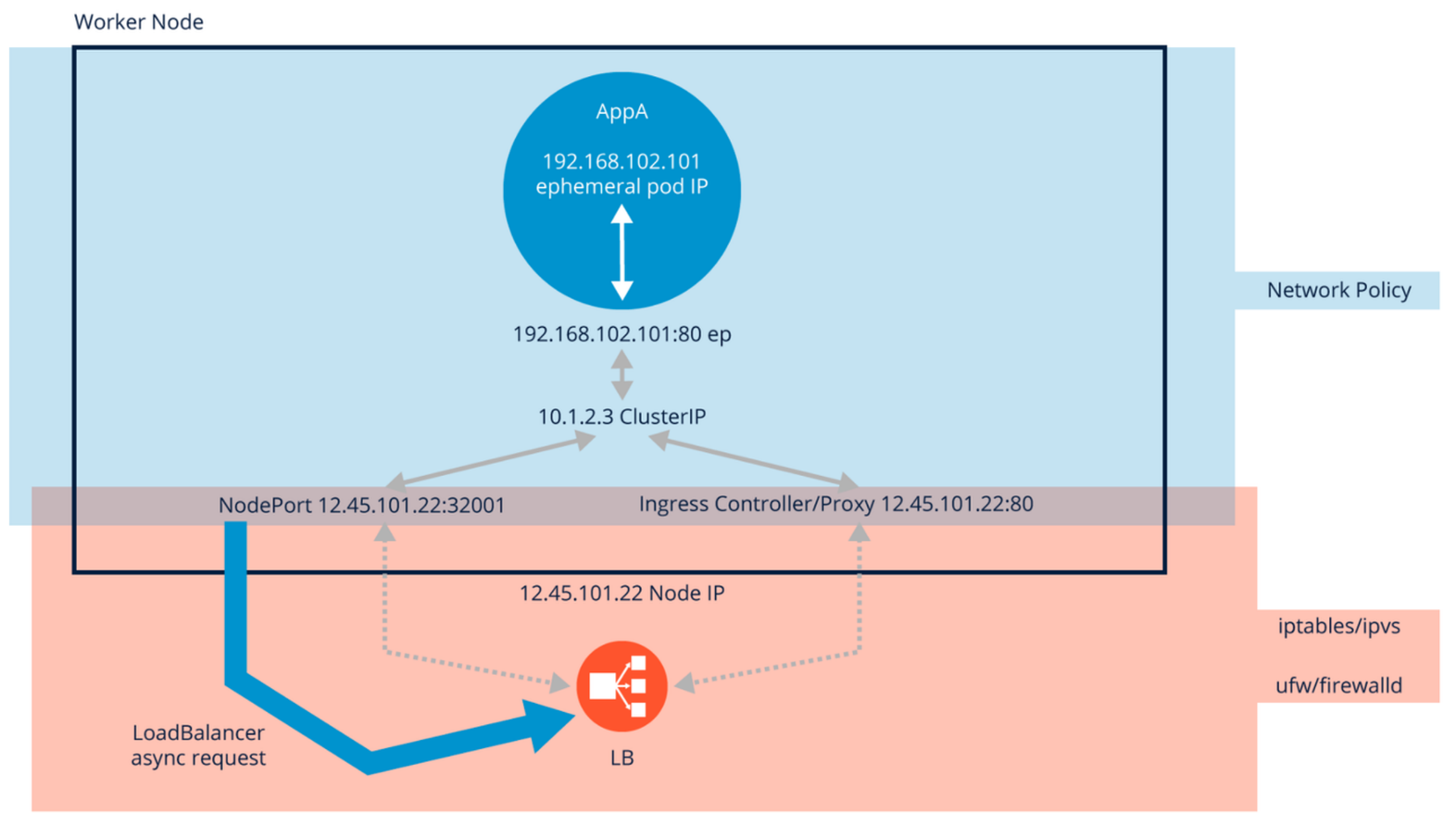
A proxy or ingress controller would then listen to the traffic and determine which service to pass along the traffic. If the pod was not on that worker node, the network plugin (such as Calico) would be used to get traffic to the correct node.

We can see the Frontend pod using a service to connect to the Backend pod. We can also see traffic leaving worker nodes to connect to storage on a Ceph cluster and authentication and authorization from an external tool such as LDAP.

Each one of these connections is a potential attack point and should be part of the ongoing monitoring and security control.

Services & Firewalls

A network-intensive environment has many areas to monitor and control. The Kubernetes architecture has been designed as a mostly open environment when part of the Borg Google project. All pods could see all nodes and vice versa.



K8s Services and Firewall Area of Effect

Services & Firewalls

In the graphic above we can see some of the components which would get traffic from outside the cluster to an AppA container running in a pod.

The top shaded area of the graphic shows that a networkpolicy would be used to manage the ingress and egress traffic for a pod.

The lower shaded area represents node, intranet, and Internet traffic controlled by an OS firewall product such as iptables, ipvs, or firewalld among others.

The kube-proxy pod will also configure iptables and ipvs rules, so care should be taken such that global network rules do not conflict with the needs of the cluster.

You may notice that the shaded areas overlap. This is because the network traffic could be effected by the operating system, as well as the Kubernetes traffic, without being aware of the other configuration.

Networking - Terms

Ingress Filtering

The filtering of data, connections, and connection requests not originating from the protected network (e.g., packets from external hosts on the Internet). It is important to filter all ingress data not specifically required for services provided by the protected network to function (e.g., a web hosting provider, HTTP port 80).

Egress Filtering

The filtering of packets originating inside the firewall. This is an important part of enforcing acceptable use policies, and reducing the risk that any malware or infected hosts can attack hosts outside the firewall (e.g., prevent malware-infected Windows zombies from participating in external DDoS attacks).

Bridging

Bridging is the act of aggregating multiple networks to appear as a single network, appearing as a direct connection of one network to the other.

Networking - Terms

Forwarding

Forwarding is the relaying of a packet from one node on a network to another. Ports can also be configured to forward on a firewall/router, allowing ingress packets on port A to be forwarded to port B on a host inside the firewall. The decision of what packet is forwarded to which port is part of a routing decision.

Network Address Translation (NAT)

Network Address Translation is the process of modifying the IP address information in the TCP headers to hide a host or network in a private network from the public network outside the firewall or router. Often, private IP addresses (RFC 1918) are used inside the firewall and translated to the public address of the firewall/router/proxy. This is often done to reduce the use of publicly addressable IPs, which are of limited quantity. NAT is not used for Kubernetes products, but may be used outside the cluster.

Stateful vs Stateless

Kubernetes works best with decoupled and transient microservices. While this model is mostly about the relationship between resources inside the cluster, it may also be used between end users and front end pods inside of Kubernetes. The less an end user is tied to a particular pod, the more flexibility and durability to pod or node failure.

Firewalls intercept traffic and use connection tracking to improve performance; this would be impacted if all connections are stateless, but the benefit may be worth the performance difference.

Many network protocols keep track of the session state. These states can be grouped into four basic categories: new, established, related, invalid:

Stateful vs Stateless

- **New** typically refers to the 3-way TCP handshake, and can be the handshake of any similar protocol.

- The majority of the traffic is likely to be under the **Established** category. Placing a rule that allows established traffic at the beginning of the ruleset can improve performance.

- **Related** is unique in that not all firewalls are capable of considering a packet as related to an established connection. Netfilter requires additional protocol-specific modules to track related traffic. FTP and VOIP require extra kernel modules to be loaded, whereas replies to DNS queries will automatically be considered as related to a known outbound DNS lookup request.

- **Invalid** traffic is that which is out of sequence according to the protocol specifications.

Stateful Packet Filtering requires more system resources as a table of connections and their relations/states is held in memory.

Anatomy of a filter

Firewalls are essentially traffic filters. As such, there are three main components of the individual rules used in a firewall to filter traffic.

Where to apply the filter is generally the first decision made when constructing a filter rule. Ingress traffic is that coming into the system or environment. Egress traffic is that generated inside the environment and leaving the local network.

Both ingress and egress traffic should be filtered at the network perimeter. Acceptable use policies can be enforced via egress filtering. Egress filtering can also ensure that local systems do not leak sensitive information and cannot attack external systems on the internet, e.g., participate in a botnet DoS, or act as spam mailers.

Deciding which traffic to filter is done by creating selection criteria and then comparing the traffic attributes to that criteria. The specific action taken is self-explanatory. These concepts apply both to operating system firewalls, as well as Kubernetes networkpolicies.

Network Plugins

While there are many network plugins to choose from, the most common in use is Project Calico. Clusters with four or five thousand nodes historically required a different plugin, as iptables-based plugins could not handle the load. Calico now supports ipvs, and remains an easy-to-use plugin with typically required security and configuration features.

The Calico Canal tool is used when the existing cluster network was Flannel, which has fewer features and does not support network policies.

Weave also supports network policies, but leverages a mesh overlay between cluster nodes, which allows for complex routing useful in complex network environments.

The Kube-router network plugin supports all of Kubernetes network policies via iptables, ipset and conntrack. All traffic is blocked by default, and must be allowed via whitelists. Service traffic is handled by ipvs and iptables.

Other plugins are Romana and kopeio.

Calico

Calico supports the Kubernetes **NetworkPolicy** object using iptables by default, at the moment. The use of ipvs is also possible, but requires some configuration prior to integration with the cluster. Calico also has its own policies which have been designed to integrate with Kubernetes in a seamless manner, and expand the functionality of network control and security features.

Calico can leverage the **WorkloadEndpoint** resource to configure how the Calico container communicates to the host. The **HostEndpoint** can be used in place of or with a **NetworkPolicy** to manage traffic. Used by itself, it will prevent all traffic by default. Through the use of declared profiles and ports, traffic can be allowed in a granular manner. This allows for the protection of the host ports in addition to pod endpoints.

The use of **GlobalNetworkPolicy** spans all namespaces. It can be used to configure connectivity rules to join workload endpoints to host endpoints in all namespaces. The global settings take precedence over other configurations such as **Profiles**, which were commonly used before **NetworkPolicy** became functional.

The Calico network plugin can also leverage an eBPF data plane, which replaces the iptables functionality and also provides a number of performance increases.

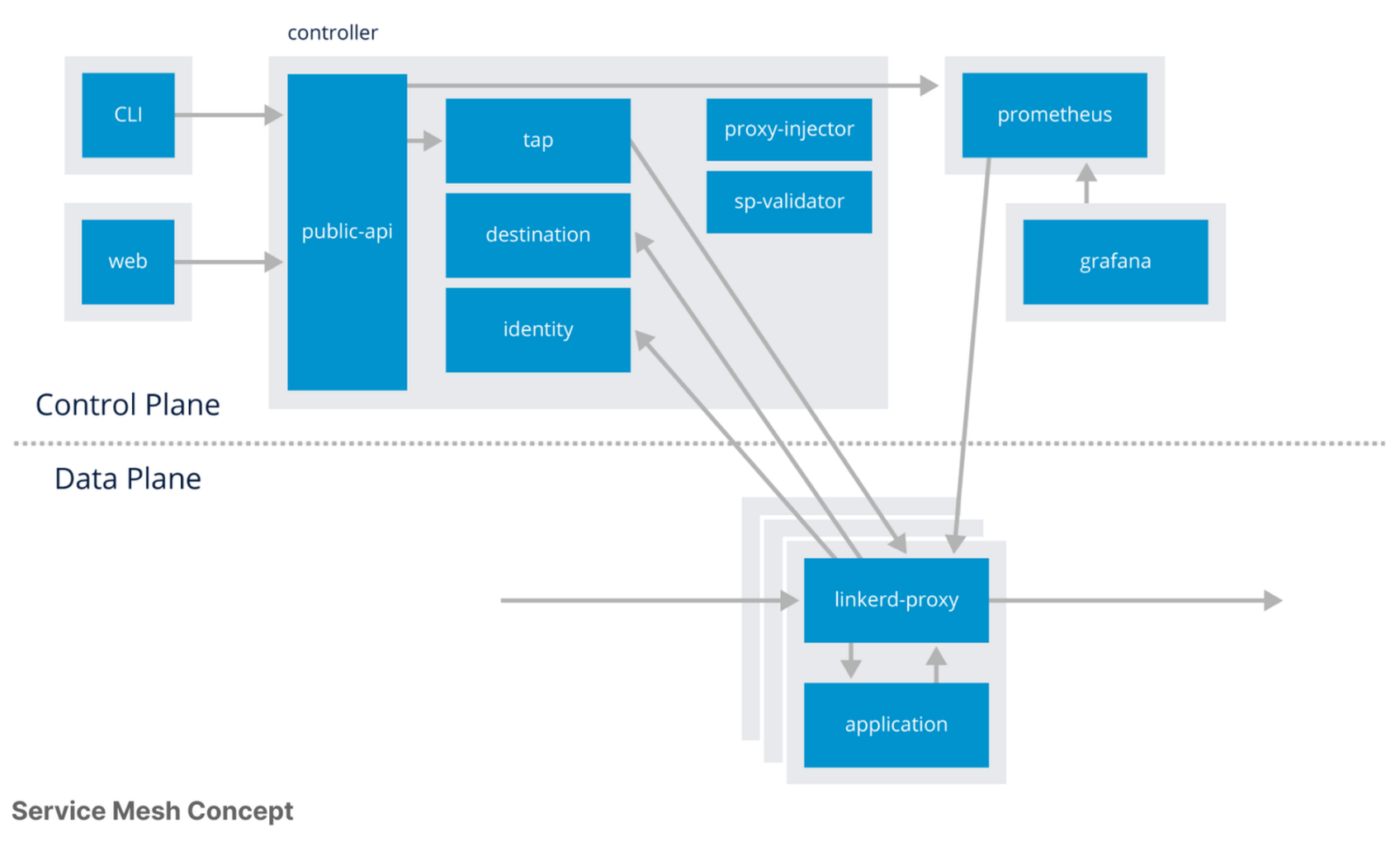
Ingress Controllers

Kubernetes is hard-coded not to allow services to listen on low number node ports. Newer releases only allow ports between 30000-32767. Should you want to expose a low port, you would need to use an ingress controller or a service mesh.

There are several ingress controllers to consider, with similar features, such as mTLS connections: Envoy Proxy, NGINX, Traefik, Ambassador. Each also allows a single point of control. Rules are added to the ingress controller to connect traffic with the proper service. Other control and security features can also be applied, with different features available for each controller.

While ingress controllers may have many features, they typically do not have a wide range of monitoring and control. Often, an extra bit of control plane is added and the term **service mesh** is applied. The term has a wide range of meaning, so there is not a strong line of division between full-featured ingress controller and service mesh.

Service Mesh



Service Mesh Concept

Service Mesh

The term service mesh is used to describe the network of microservices that make up such applications and the interactions between them. As a service mesh grows in size and complexity, it can become harder to understand and manage. Its requirements can include discovery, load balancing, failure recovery, metrics, and monitoring. A service mesh also often has more complex operational requirements, like A/B testing, canary rollouts, rate limiting, access control, and end-to-end authentication.

A service mesh is a configurable, low-latency infrastructure layer designed to handle a high volume of network-based interprocess communication among application infrastructure services using application programming interfaces (APIs). A service mesh ensures that communication among containerized and often ephemeral application infrastructure services is fast, reliable, and secure. The mesh provides critical capabilities including service discovery, load balancing, encryption, observability, traceability, authentication and authorization, and support for the circuit breaker pattern.

Service Mesh

The service mesh is usually implemented by providing a proxy instance, called a sidecar, for each service instance. Sidecars handle interservice communications, monitoring, and security-related concerns – indeed, anything that can be abstracted away from the individual services. This way, developers can handle development, support, and maintenance for the application code in the services; operations teams can maintain the service mesh and run the app.

There are several service mesh options, with more coming to market, and doing the same basic features.

- Istio
- Linkerd
- Contour
- Aspen

Network Policies

Many of the popular network plugins support the use of NetworkPolicies, which provide an inside-the-cluster firewall. The original Kubernetes architecture called for all pods to see and be seen by all nodes. In a multi-tenant environment, you may want of allow some traffic and deny other. A network policy allows this to happen.

NetworkPolicies are divided into ingress and egress rules. If you declare a direction of transport, then only the traffic explicitly allowed will connect. As a result, a minimal YAML file can be used to create an all-closed environment.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-all
  namespace: prod-ns
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Network Policies

The use of curly brackets {} as a pod selector is not a null value, but rather means all pods. Also, the use of except statements can create more granular network configurations.

While network policies have been in use for several releases, they are not as feature-filled as traditional firewalls such as iptables or firewalld. You may end up with a combination of products providing protection between objects. It is important to document who is responsible for maintaining multiple products.

Network Policies

```
....
spec:
  podSelector:
    matchLabels:
      app: scooter
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: scooter
          role: search
    - podSelector:
        matchLabels:
          app: scooter
          role: api
    - podSelector:
        matchLabels:
          app: inventory
          role: web
```

Network Policies

In this example, we see more than one podSelector in use. These values are treated as or statements, meaning they all combine to create a list of allowed selectors.

This whitelist applies to pods that have two labels, app: scooter and role: db. Those pods will receive inbound traffic from pods with any of the following matching labels.