Kubernetes

On-Prem or Cloud Agnostic

Overview

Kubernetes topic	Technology
Installing Kubernetes on premise	kubeadm, RKE
Managing SSL	cert-manager
Service Mesh, LB and Proxy	Envoy, Istio
Networking	Calico
Secret store	Vault

Objectives

- To be able to use Kubernetes on-prem or in a cloud agnostic way
  - This allows you to use Kubernetes in an enterprise environment
- After this course you should be able to deploy Kubernetes anywhere
  - using your own integrations
  - like storage, certificates, authentication and so on

kubeadm

- kubeadm is a toolkit by Kubernetes to create a cluster
- It works on any deb/rpm compatible Linux OS, for example Ubuntu, Debian, RedHat or CentOS
  - This is the main advantage of kubeadm, because a lot of tools are OS / Cloud specific
- It's very easy to use and lets you spin up your Kubernetes cluster in just a couple of minutes
- kubeadm supports bootstrap tokens
  - Those are simple tokens that can be used to create a cluster or to join nodes later on
  - The tokens are in the format abcdef.0123456789abcdef
- kubeadm supports upgrading / downgrading clusters
- It does not install a networking solution
  - You'll have to install a Container Network Interface - compliant network solution yourself using kubectl apply

cert-manager

- If you want to use a secure http connection (https), you need to have certificates
- Those certificates can be bought, or can be issued by some public cloud providers, like AWS's Certificate Manager
- Managing SSL / TLS certificates yourself often takes a lot of time and are time consuming to install and extend
  - You also cannot issue your own certificates for production websites, as they are not trusted by the common internet browsers (Chrome, IE, …)
- Cert-manager can ease the issuing of certificates and the management of it
- Cert-manager can use letsencrypt

- Let’s encrypt is a free, automated and open Certificate Authority
    - Let’s encrypt can issue certificates for free for your app or website
    - You’ll need to prove to let’s encrypt that you are the owner of a domain
    - After that, they’ll issue a certificate for you
    - The certificate is recognized by major software vendors and browsers
- Cert-manager can automate the verification process for let’s encrypt
- With Let’s encrypt you’ll also have to renew certificates every couple of months
- Cert-Manager will periodically check the validity of the certificates and will start the renewal process if necessary

- Let’s encrypt in combination with cert-manager takes away a lot of hassle to deal with certificates, allowing you to secure your endpoints in an easy, affordable way
- You can only issue certificates for a domain name you own
- You’ll need to have a domain name like xyz.com
    - You can get one for free from www.dot.tk or other providers
    - Or, you can buy one through namecheap.com / AWS route53 / any other provider that sells domain names
    - Less popular extensions only cost a few dollars
- https://eric-v-documentation.readthedocs.io/en/latest/onprem-cloudagnostic-k8s.html#demo-cert-manager

Istio - Envoy

- When you break up a monolith application (1 codebase), into micro-services (multiple codebases), you end up with lots of services that need to be able to communicate with each other
- These communications between services need to be able to be fast, reliable and flexible
- To be able to implement this, you need a service mesh
    - A service mesh is an infrastructure layer for handling these service-to-service communications
    - This is usually implemented using proxies
    - Proxies manage these communications and ensure they're fast, reliable and flexible

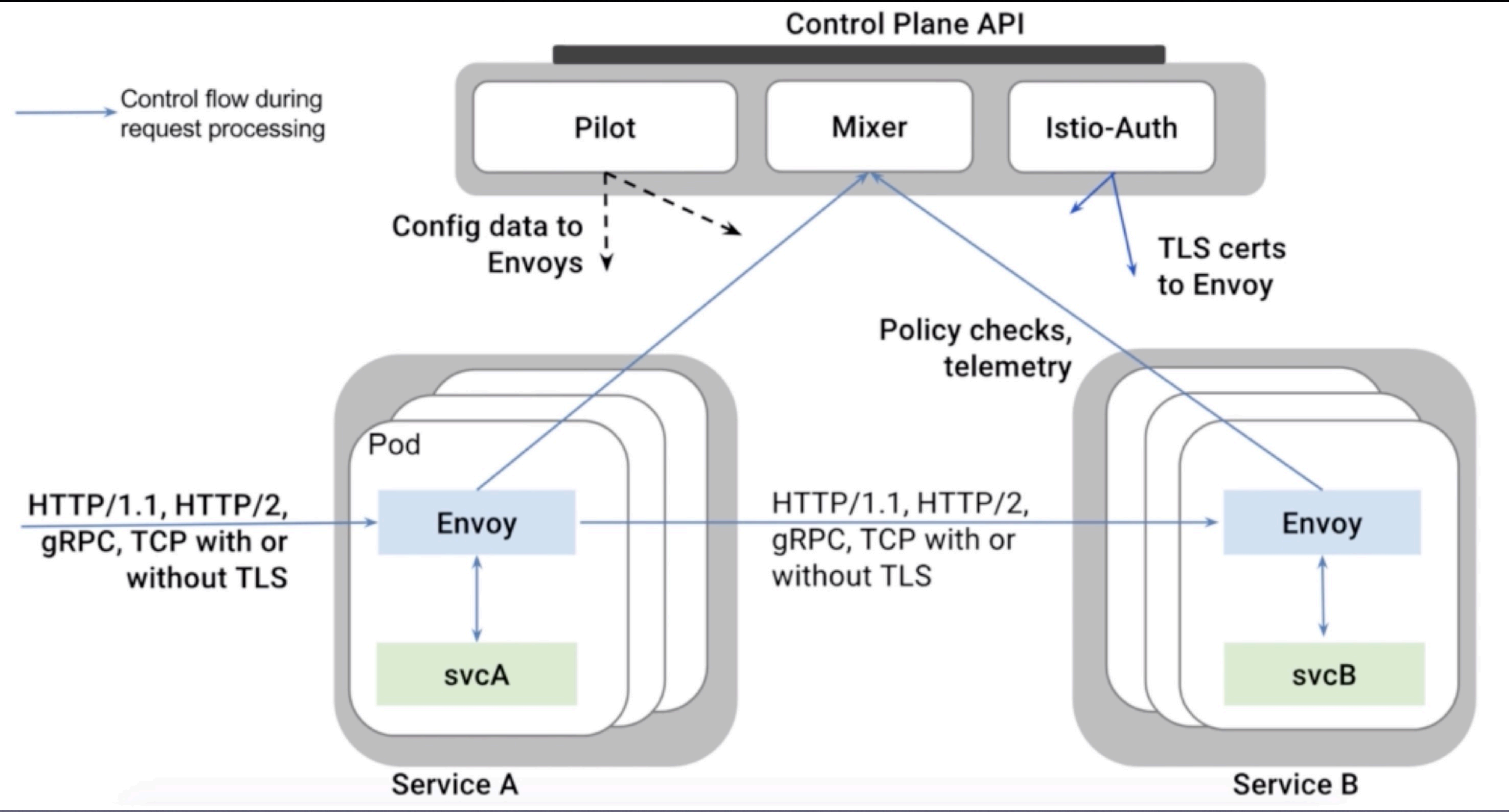
- Envoy is a such a proxy
    - It is designed for cloud native applications
- Was originally built at Lyft
- Envoy is a High Performance distributed proxy written in C++
- You can see it as an iteration of the NGINX, HAProxy, hardware / cloud load balancers
- It’s comparable with Linkerd
    - While there’s a lot of overlap, each solution has its own distinct features

- Envoy Features
  - Small memory footprint
  - HTTP/2 and gRPC support
    - It’s a transparent HTTP/1.1 to HTTP/2 proxy
      - Not all browsers support HTTP/2 yet, so incoming requests can be HTTP/ 1.1, but internally requests can be HTTP/2
  - Advanced Loadbalancer Features (automatic retries, circuit braking, rate limiting, request shadowing, zone load balancing, …)
  - Configuration can be dynamically managed using an API
  - Native support for distributed tracing

- Comparison to linkerd
    - Linkerd has more features, but that comes at a price of higher cpu and memory footprint
        - Linkerd is built on top of Netty and Finagle (JVM based), whereas Envoy is written in C++
        - If you're looking for more features, you might want to look at Linkerd, if you're looking for speed and low resource utilization, Envoy wins
            - Istio, discussed next, can give you the best of both worlds
    - Linkerd integrates with Consul and Zookeeper for service discovery
    - Envoy supports hot reloading using an API, Linkerd does not (by design)

Istio

- Istio is an open platform to connect, manage, and secure microservices (Definition: https://istio.io/docs/concepts/what-is-istio/overview.html)
- Key capabilities include:
  - It supports Kubernetes
  - Can control traffic between services, can make it more robust and reliable
  - Can show you dependencies and the flow between services
  - Provides access policies and authentication within your service mesh


Control Plane API
Control flow during request processing
Pilot
Mixer
Istio-Auth
Config data to Envoys
TLS certs to Envoy
Policy checks, telemetry
Pod
HTTP/1.1, HTTP/2, gRPC, TCP with or without TLS
Envoy
svcA
HTTP/1.1, HTTP/2, gRPC, TCP with or without TLS
Envoy
svcB
Service A
Service B

Istio Components

- Envoy (data plane)
    - Istio uses the Envoy proxy in its data plane
    - It uses a sidecar deployment, which means a deployment along the application (a one to one relation between app/pod and proxy)
- Mixer (control plane)
    - Responsible for enforcing access control and usage policies
    - Collects telemetry data from Envoy
- Pilot (control plane)
    - Responsible for service discovery, traffic management and resiliency
        - A/B tests and canary deployments
        - Timeouts, retries, circuit brakers
    - It does this by converting Istio rules to Envoy configurations
- Istio Auth (control plane)
    - Service-to-service and end-user authentication using mutual TLS

Calico

- Calico provides secure network connectivity for containers and virtual machine workloads. (Definition: https://docs.projectcalico.org/v3.1/introduction/)
- Calico is a Software Defined Network, with a simplified model, with cloud-native in mind
- Calico creates a flat Layer 3 network using BGP (Border Gateway Protocol) as routing mechanism
  - BGP is also used as the “internet routing protocol” to route between providers (it’s a proven, scalable technology)
- Policy driven network security using the Kubernetes Network Policy API
  - Fine-grain control over the network, using the same Kubernetes API (using yaml files) as you’re used to

- Only use overlay if necessary, reducing overhead and increasing performance
  - An overlay network does IP encapsulation, but often those IP packets can be routed without adding those extra headers to IP packets
- Works with Kubernetes, but also with OpenStack, Mesos, and others
- Uses etcd as backend (Kubernetes also uses etcd - a distributed key value store using Raft consensus)
- Works on major cloud providers like AWS, GCE (also Kubernetes Engine), Azure (ACS)
  - Will also support the hosted kubernetes services AWS EKS and Azure AKS when they'll be GA
- Works well within enterprise environments
  - Either without overlay
  - With IP-in-IP tunneling
  - Or using an overlay (VxLAN) network like Flannel

Calico components

- Calicoctl
  - Allows you to manage the Calico network and security policy
- Felix
  - Daemon that runs on every machine (calico-node DaemonSet)
  - Responsible for
    - programming routes and ACL on the nodes itself
    - Interface management (interacts with kernel - think about MAC address / IP level configuration)
  - Reports on health and state of the network
- BGP Client (BIRD)
  - Runs next to Felix (still within the calico-node DaemonSet)

- Reads routing state that Felix programmed and distributes this information to other nodes
  - Basically that's what BGP needs to do, it needs to make the other nodes aware of routing information to ensure traffic is efficiently routed
- BGP Route Reflector
  - All BGP clients are connected to each other, which may become a limiting factor
  - In larger deployments, a BGP route reflector might be setup, which acts as a central point where BGP clients connect to (instead of having a mesh topology)
- Once Calico is setup, you can create a network policy in Kubernetes

- You can first create a network policy to deny all access to all pods (then afterwards you can open the ports that are needed):

    -
      ```
      apiVersion: networking.k8s.io/v1
      kind: NetworkPolicy
      metadata:
        name: deny-all
        namespace: app
        spec:
          podSelector:
            matchLabels: {}
      ```

- At this point the pods are isolated, you'll not be able to connect from one pod to another anymore

- Isolated vs non-isolated

    - By default pods are non-isolated

        - Pods accept traffic from any source

- By having a network policy with a selector that selects them (the previous one selects all pods), network access is denied by default

    - The pod now becomes isolated

    - Only connections that are defined in the network policy are allowed

    - This is on a namespace basis

- You can now add a new rule to enable network access to a pod:

-
  ```
  apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-my-app
    namespace: apps
  spec:
    podSelector:
      matchLabels:
        app: my-app
    ingress:
      - from:
        - podSelector:
            matchLabels:
              app: a-pod
  ```

- https://eric-v-documentation.readthedocs.io/en/latest/onprem-cloudagnostic-k8s.html#demo-calico-example

Vault

Managing credentials in a distributed environment

- Vault is a tool for managing secrets
    - For example: passwords, API keys, SSH keys, certificates
- It’s opensource and released by HashiCorp (like Vagrant, terraform, and other well known tools)
- Some use cases are:
    - General Secret Storage
    - Employee Credential Storage (Sharing credentials, but using audit log, with ability to roll over credentials)
    - API key generation for scripts (Dynamic Secrets)
    - Data Encryption / Decryption

Vault features

- Secure Secret Storage
  - Encrypted key-value pairs can be stored in Vault
- Dynamic Secrets
  - Vault can create on-demand secrets and revoke them after a period of time (when the client lease is up)
  - For example AWS credentials to access an S3 bucket
- Data Encryption
  - Vault can encrypt / decrypt data without storing it
- Leasing and Renewal
  - Secrets in Vault have a lease (a time to live)
  - When the lease is up, the secret will be revoked (deleted)
  - Clients can ask for a renewal (a new secret) using an API

- Revocation
    - Easy revocation features
    - For example, all secrets of a particular user can be removed
- In April 2018 CoreOS released the Vault Operator
- It allows you to easily deploy Vault on Kubernetes
- It allows you to configure and maintain Vault using the Kubernetes API (using yaml files and kubectl)
- It gives you a good alternative to secret management tools on public cloud (like the AWS Secrets Manager or AWS Parameter store)
- https://eric-v-documentation.readthedocs.io/en/latest/onprem-cloudagnostic-k8s.html#demo-vault