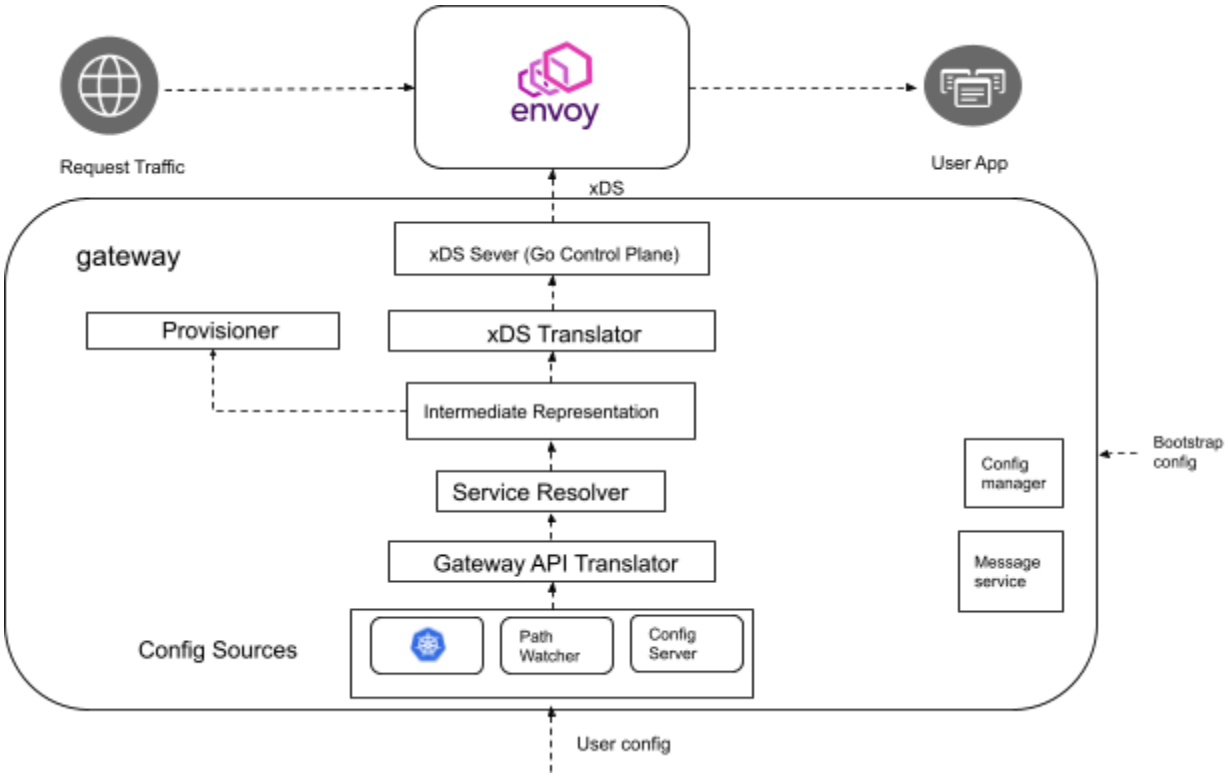

Gateway Configuration

- User Config - This configuration will be based on the Gateway API and will provide:
    - Infrastructure Management capabilities for the Infrastructure Administrator to provision the infrastructure required to run Envoy.
    - Ingress and API Gateway capabilities for the application developer to define networking and security intent for their incoming traffic.
- Bootstrap Config - This is the configuration provided by the Infrastructure Administrator that allows them to configure various aspects of envoy gateway. This is similar to what Envoy has today .

Gateway Components

- Config Sources - This component is responsible for consuming the user configuration from various platforms. It will also aggregate resources from multiple individual config sources before publishing them to the translation layers. Data persistence should be tied to the specific config source's capabilities. For e.g. in Kubernetes, the resources will persist in etcd, if using the path watcher, the resources will persist in a file.

    - Kubernetes - The Kubernetes controller is responsible for watching the Kubernetes API Server for Gateway resources, fetches them, and publishes it to the next component for further processing.
    - Path Watcher - This components watches for file changes in a path, allowing the user to configure the Envoy Gateway using configurations saved in a file
    - Config Server - This is a HTTP/gRPC Server allowing the Envoy Gateway to be configured from a remote endpoint, similar to Envoy's xDS Server.

- Intermediate Representation (IR) - This is an internal data model that user facing APIs are translated into allowing for internal services & components to be decoupled.

- Config Manager - This component consumes the Bootstrap Config, and spawns the appropriate services in envoy gateway based on the config. It could be based on design patterns such as this.

- Message Service - This component allows internal services to publish message / data types as well as subscribe to them. A message bus architecture allows components to be loosely coupled, work in an asynchronous manner and also scale out into multiple processes if needed. Here is an example of a message service using go channels.

- Service Resolver - This component preprocesses the IR resources and resolves the services into endpoints providing more precise load balancing and resilience policies. For e.g. in Kubernetes, a controller service could watch for EndpointSlice resources, converting Services to Endpoints, allowing for Envoy to skip kube-proxy's load balancing layer. This component is tied to the platform where it is running. This Is an optional component that can be disabled to allow the services to be resolved by the underlying DNS resolver or by explicitly specifying static IPs.

- Gateway API Translator - This is a platform agnostic translator that translates Gateway API resources to an Intermediate Representation. This is useful for decoupling the internal services from the user API layer.

- Envoy Translator - This component translates the IR into Envoy xDS Resources.

- xDS Server - This component is a xDS gRPC Server based on the Envoy Go Control Plane project that implements the xDS Server Protocol and is responsible for configuring Envoy resources in the Envoy proxy.

- Provisioner -
    - Envoy - provisions a Envoy based Load balancer service . This is a platform specific component. For example, a Terraform or Ansible provisioner could be added in the future to provision the Envoy infra in a non-k8s env.

    - Auxiliary Control Planes - These components are responsible for handling out of band control plane traffic & decisions sent by Envoy.
        - Rate Limit service - This is based on the Envoy Rate Limit Service and will consume the IR and translate it into the server side rate limiting config. A similar envoy translator sub component would translate the IR into Envoy's ratelimit filter.

Design Decisions

- Will Envoy Gateway manage a fleet of Envoy proxies with different configurations (xDS as well as Bootstrap) ? Yes, EG will manage envoy resource config per envoy gateway ID.
- Will Envoy Gateway in Kubernetes be scoped to watching a subset of namespaces/labels in Kubernetes ? Could be
- Will Envoy Gateway in Kubernetes be scoped to resolving service endpoints for a subset of upstream services in Kubernetes ? Could be
- Where will Contour & Emissary translators fit ? Out of repo translators would translate their native API into the Gateway API for now. If this is not feasible, EG will be need to provide extension points within EG .
- What is in scope for Native Gateway API Extensions ? - EnvoyPatch / xDS, Ratelimiting , OIDC, Authn, Authz, TLS Issuer, Load balancing, Caching, Resiliency / Outlier Detection / Retries ?
- How do vendors deal with non native Gateway API extensions ? Does the Aux Control Plane / Provisioner need to be extendible ? Does the Envoy Translator need to be extendible ? Skipping this for now
- Is Service Resolver a key component in the design ? Answered in Service resolver section
    - 1. Should the resolver support Endpoints and EndpointSlice types?
    - 2. Should the resolver support all EndpointSlice address types?
    - 3. Should other EndpointSlice fields be considered by the resolver, eg hints, hostname, nodeName, etc.
    - xref: https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.23/#endpoint-v1-discovery-k8s-io
- Will the project support multiple Config Sources ? Answered in Config Source section.
- With the goal of EG supporting multiple environments and not just k8s, what are the plans to persist data? Answered in Config Source section

Feedback

1. Thank you arko@tetrate.io for putting this doc together. I'm working on this diagram to express my ideas related to the design. It appears that we're on the same page with many design aspects so we're off to a good start. I'm interested in getting input from the community so we can iterate.