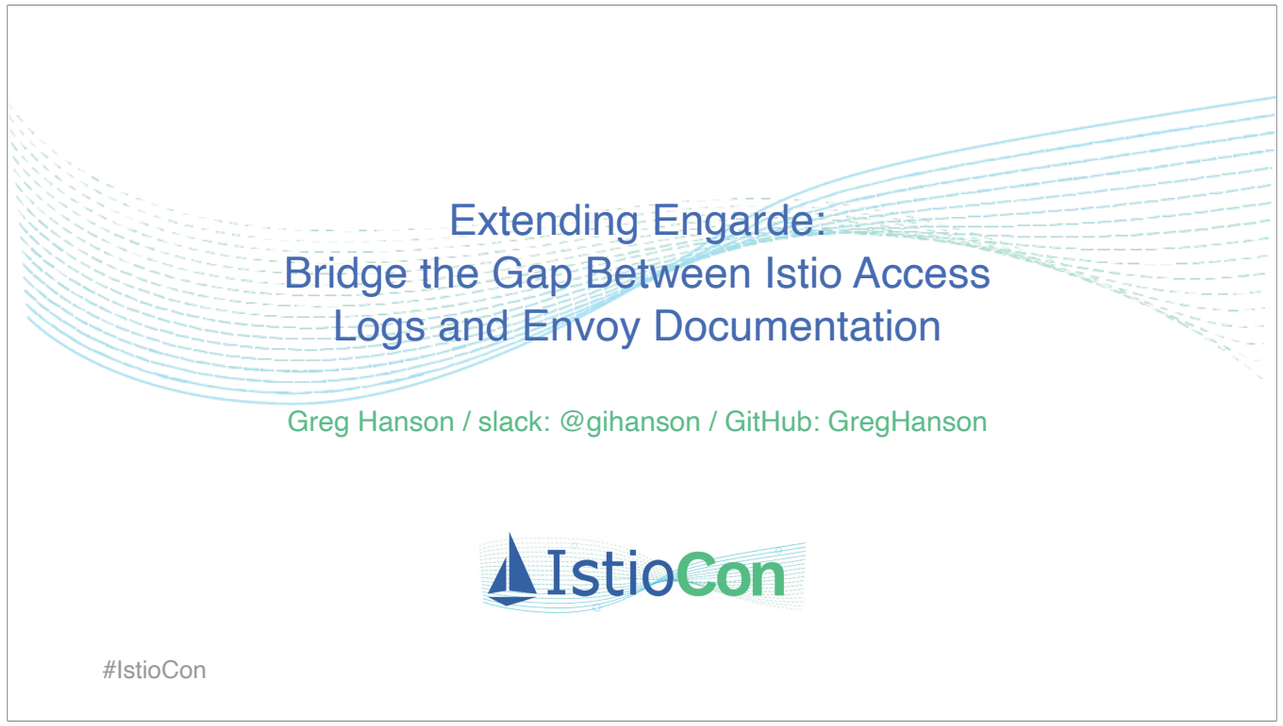


Hello everyone. My name is Greg Hanson. I am a Software developer at IBM and I have been working on the Istio project since it started. First time giving a lightning talk, so here we go!

So what is Istio? Just kidding, it’s the end of the week and the conference has the same name - so I think I can pretty safely skip over this slide.

One of my areas of expertise in Istio is understanding how various resources generate the appropriate configurations which are supplied to the envoy process running in every Istio-proxy sidecar in the form of LDS/RDS/CDS/etc. This means that as a part of my role at work, most of the questions that get directed my way are along the lines of “I created this VirtualService, DestinationRule or ServiceEntry and my requests are failing”.

What’s the first thing I ask for? Access Logs

Access Logs

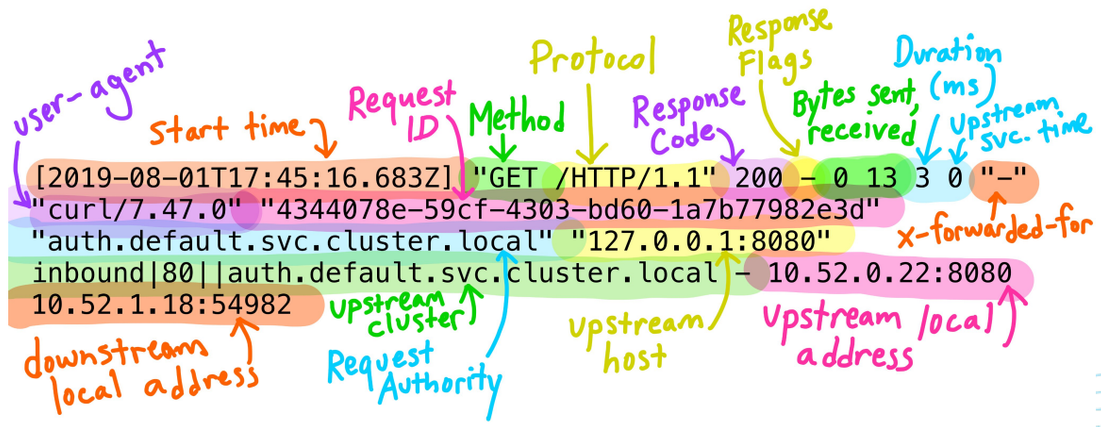
[2021-02-11T21:57:47.658Z] "GET /headers HTTP/1.1" 200 - "-" 0 551 22 22 "-" "curl/7.69.1" "492f0cf3-e6e4-9182-b7c4-ee9d2ec06cfc" "httpbin:8000" "172.30.213.81:80" outbound|8000||httpbin.default.svc.cluster.local 172.30.213.80:43640 172.21.92.90:8000 172.30.213.80:46244 - default

That’s right, access logs. As a testament to how long I have been working with Istio, but long before the days of the istioctl, which provided a means to inspect envoy’s configuration for Listeners , Routes , or Clusters - there were only access logs and config_dumps. If you can step your way through an envoy config_dump from one of Istio’s sidecars, props to you. Find me on slack, lets be friends.

Now, I could ask to see their VirtualService or their DestinationRule first , after all those types of errors can be briefly summarized into the following categories

- connection errors with your ServiceEntry? Usually a port or protocol mismatch
- 404’s ? probably something wrong with the hostname or VirtualService
- 500’s ? probably a bad DestinationRule

All that hidden information



but if their applications are using an unknown hostname, a different port, or maybe there is another ServiceEntry with “protocol: TCP” port definition capturing more traffic than intended - looking at their rules is not going to tell the whole story. In cases of failed requests for sidecar-to-sidecar traffic within the mesh, or between sidecar and the egress gateway as it attempts to communicate with an external service - access logs can help give a better idea of where in the request path an error occurs.

- Is that 404 coming from the outbound proxy at the source or the inbound proxy at the destination? Did it even reach the destination?
- For that 503 failure, does the cluster name match with the expected service name and the subset you have defined in your DestinationRule?
- Also, is the ServiceEntry, VirtualService or DestinationRules even impacting the request traffic or did the request fall through to the allow_any PassThroughCluster?

These access logs provide an extensive amount of information that can be used to troubleshoot issues. The error could be:
- how Istio is capturing the traffic, at the source or destination
- how traffic is routed using the user created traffic management resources
- Or how the application itself is attempting to communicate with resources inside and outside the mesh.

The logs also encourage novice Istio users to start learning to interpret and diagnose errors themselves, no more hand holding!! From an access log entry, users know which Listener, Route, and Cluster in Envoy was used during the request - and you can use istioctl to inspect the json for those objects to see if there are any misconfigured hostnames, timeouts, or TLS contexts - and it even says which routing resource was responsible to generating the configuration.

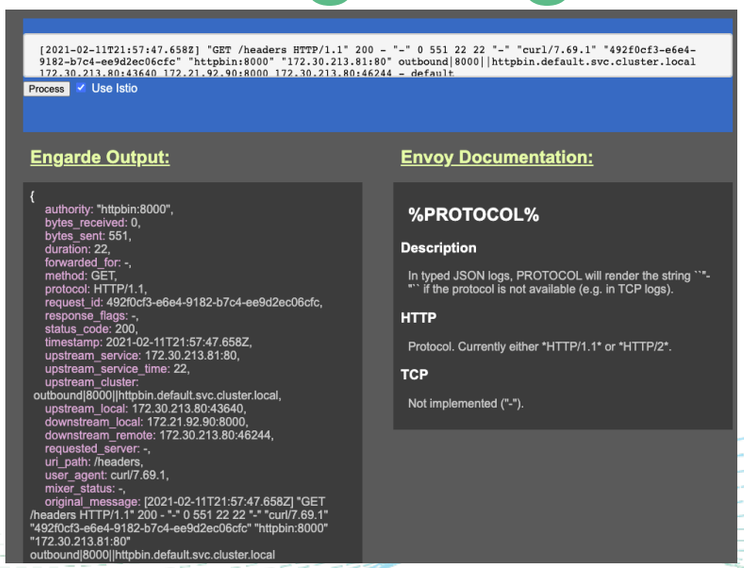
However, most people don’t know what those the access log fields mean. It used to be that if I wanted someone to be able to figure this by themselves, I would have to point users to the line in istio source code which defines the custom log format, and then provide a link to the envoyproxy.io documentation to explain what those fields mean. The process is a little convoluted at this point and only really ambitious debuggers would bother jumping through those hoops.

Introducing Engarde


```
{
  "authority": "httpbin:8000",
  "bytes_received": "0",
  "bytes_sent": "551",
  "duration": "22",
  "forwarded_for": "-",
  "method": "GET",
  "protocol": "HTTP/1.1",
  "request_id": "492f0cf3-e6e4-9182-b7c4-ee9d2ec06cfc",
  "response_flags": "-",
  "status_code": "200",
  "timestamp": "2021-02-11T21:57:47.658Z",
  "upstream_service": "172.30.213.81:80",
  "upstream_service_time": "22",
  "upstream_cluster": "outbound|8000||httpbin.default.svc.cluster.local",
  "upstream_local": "172.30.213.80:43640",
  "downstream_local": "172.21.92.90:8000",
  "downstream_remote": "172.30.213.80:46244",
  "uri_path": "/headers",
  "user_agent": "curl/7.69.1",
  "mixer_status": "-",
  "original_message": "[2021-02-11T21:57:47.658Z] \"GET /headers HTTP/1.1\" 200
\"httpbin:8000\" \"172.30.213.81:80\" outbound|8000||httpbin.default.svc.cluster
}
```


Then at KubeCon North America in 2019, I heard about Engarde. If you have heard about it already - great, just bear with me. Take an access log entry, pipe it into the Engarde tool for processing, add in jq, and you get a pretty printed json representation of the log, complete with labeled field names. It even included a special flag just for parsing Istio's custom log format. So I thought, ha-hah! Smarter debuggers means less pouring through access logs for me. But it wasn't quite there yet. After all, when someone asks me "why is there a URX response code for all these failures, what does that mean?" The result, I still have the manual step of pointing to the envoyproxy.io access log page. It's gotten to the point where if I type in envoy proxy in my browser , it knows to send me directly to the access log page instead of the main documentation page.

Extending Engarde



So what can I do to make to make it include everything I want in one place? After some HTML and CSS educational refreshers, I wrote (or hacked together - please be gentle) a UI which incorporates Engarde and Envoy's documentation all into one place. Paste in the access log entry (using Istio's or Envoy's default log format), hit the button, and out comes the processed json output from Engarde. Nothing new at this point, just a UI for a CLI tool.

So what I did was I wrote a scraper for Envoy's documentation, pulling everything regarding access log command operators and their corresponding descriptions. From the UI, if you click on one of the json field names or values, you get the description of the fields pulled directly from the documentation. Everything in one place - hooray! I'll hold for applause. __ I developed this mostly to make my personal workday easier but I hope someone else will find it as helpful as I do!

Thank you!

Engarde:

- https://github.com/nitishm/engarde

Code for the UI Extension

- https://github.com/GregHanson/engarde-viewer

@gihanson on Istio Slack

#IstioCon

Special thanks to the creator of Engarde, you are awesome and and clearly my inspiration.

Also, thanks to Envoy for having docs that a beginner like myself was able to get a scraper working pretty easily

The code for my UI and instructions for how to use can be found as a repository on GitHub

Just in case I did not stress it enough earlier - I am a beginner in html and css so I welcome any suggestions or improvements. Also, always curious if there are any other feature suggestions that could round out the tool.

And of course, feel free to reach out to me on Istio slack as @gihanson.

Thanks everyone! Happy Service Mesh-ing!