Moving Cloud Native Beyond HTTP

Adding protocols to unlock new use cases

Hi everyone. In this session we'll be discussing protocols, specifically networking protocols. I'll cover some of the challenges we face deploying protocols to production and where we as a community can make it easier. Let's go.

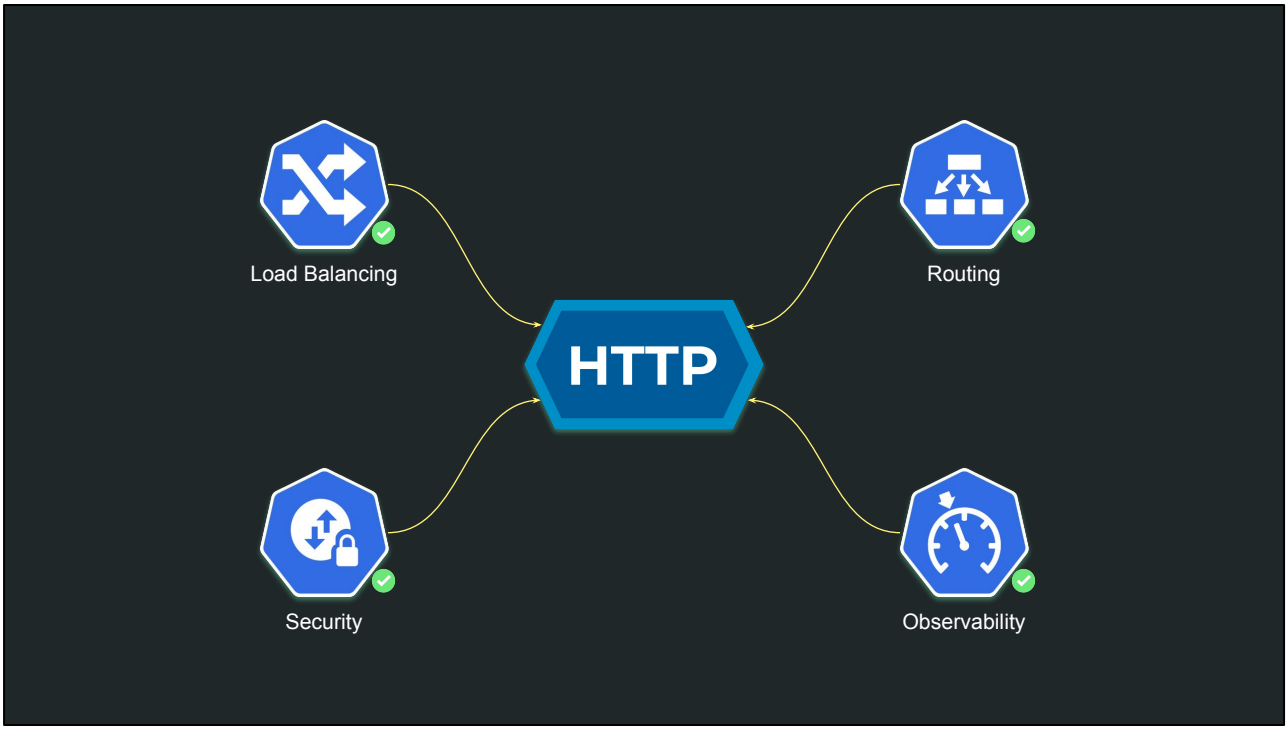
Jonathan Beri

Golioth, an IoT startup

@beriberikix

My name is Jonathan Beri and I work on an IoT startup. You can find me on twitter at beriberikix if you want to chat about protocols, or anything else!

We should probably begin our discussion with HTTP, as it is the networking protocol most people are familiar with. It’s the lingua franca of the web and a large percentage of web services are built using HTTP.



Now once you have your HTTP service you're not done - you need a bunch of things to deploy it to production and in the cloud. Things like load balancing, routing, security and observability. Since http is such a popular protocol,

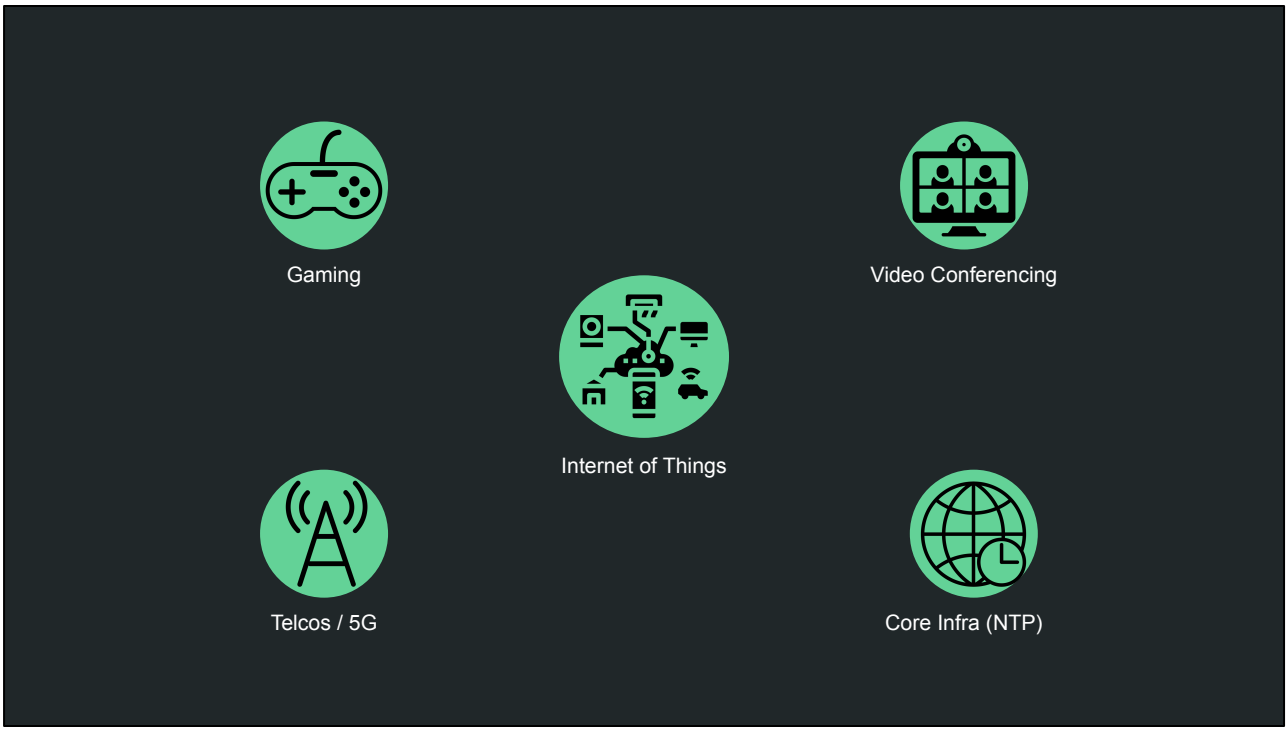
[animate]

different projects and operators provide these things for HTTP out of the box. Which is great, since you would otherwise have to build these things yourself.

But HTTP isn't the only protocol you use all the time.

There’s also DNS, everyone’s favorite service. You use it when you want to configure a server to map to a domain name or enable services within a cluster to discover each other. But what if you wanted to *create* a DNS service? Turns out that this is thing people want to do. Well, you may not have realized that DNS is actually a suite of protocols and when you use services like Amazon Route 53 or CoreDNS, those are actually implementations of DNS protocols.

DNS, like HTTP, has similar things it needs in order to be deployed to production. However, since DNS is a less common protocol compared to HTTP, it isn't supported out of the box by many of the cloud native projects, and therefore you have to build a lot of these things for DNS yourself. And that makes it way harder to build your own solution that implements DNS.



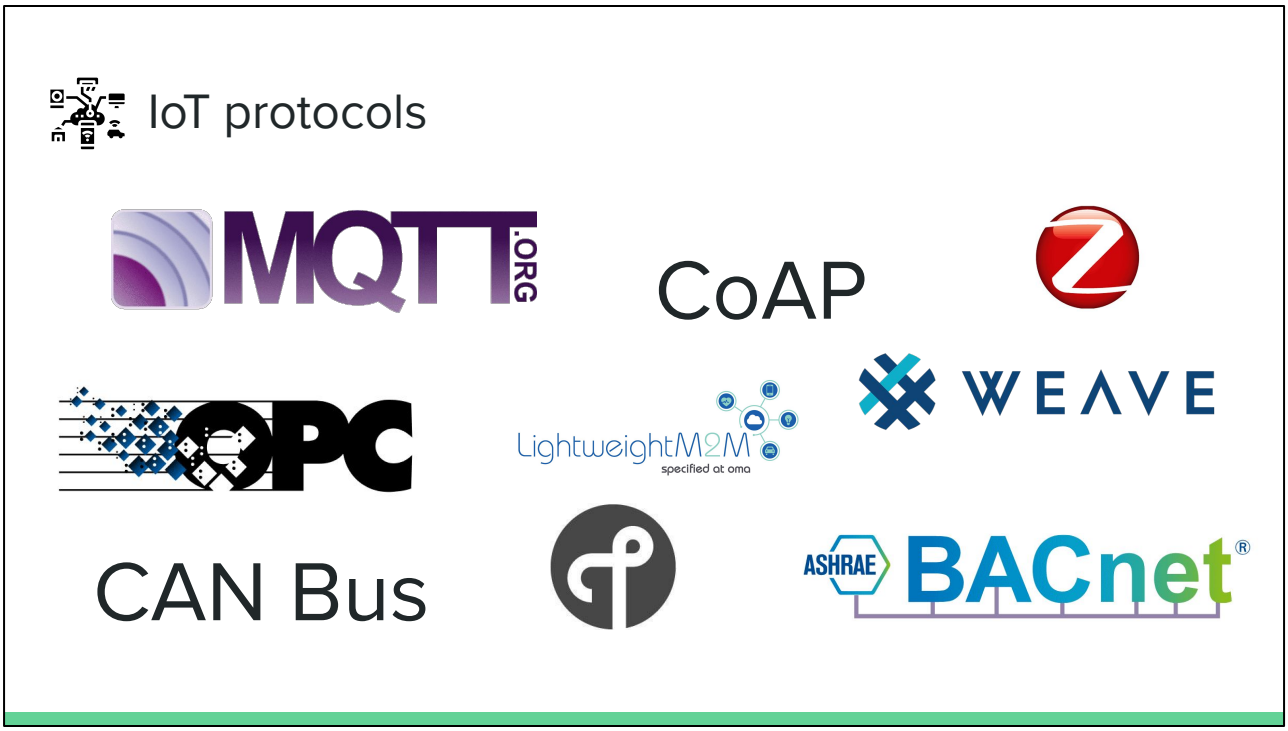
And there a lot of application and industry protocols that are designed fill a specific a purpose. I've listed a few examples here but they range from synchronizing game state to streaming WebRTC to the internet of things. And actually, IoT

[animate]

is what first got me interested in exploring the topic of implementing protocols in the first place.

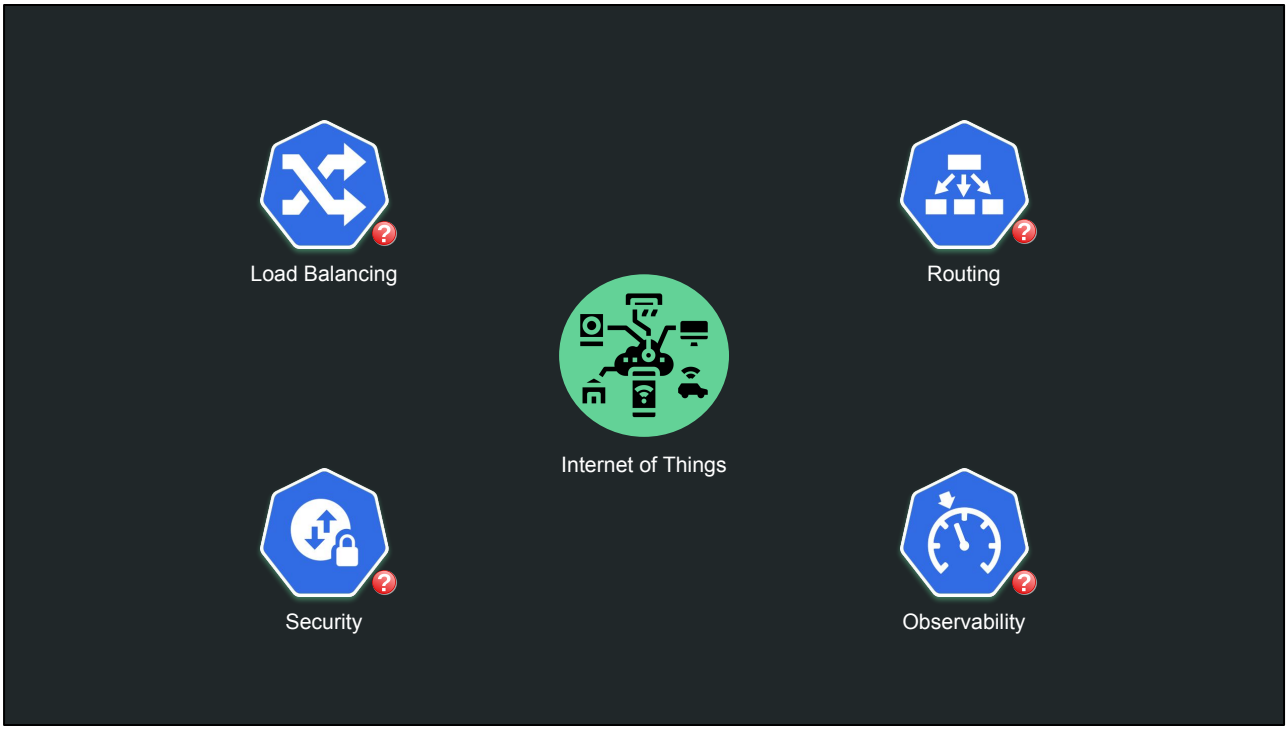
Attribution:

https://thenounproject.com/search/?q=gamepad&i=665185
https://thenounproject.com/search/?q=video+conferencing&i=3364005
https://thenounproject.com/search/?q=cellular&i=311763
https://thenounproject.com/search/?q=network+time&i=2180671
https://thenounproject.com/search/?q=internet+of+things&i=3379440

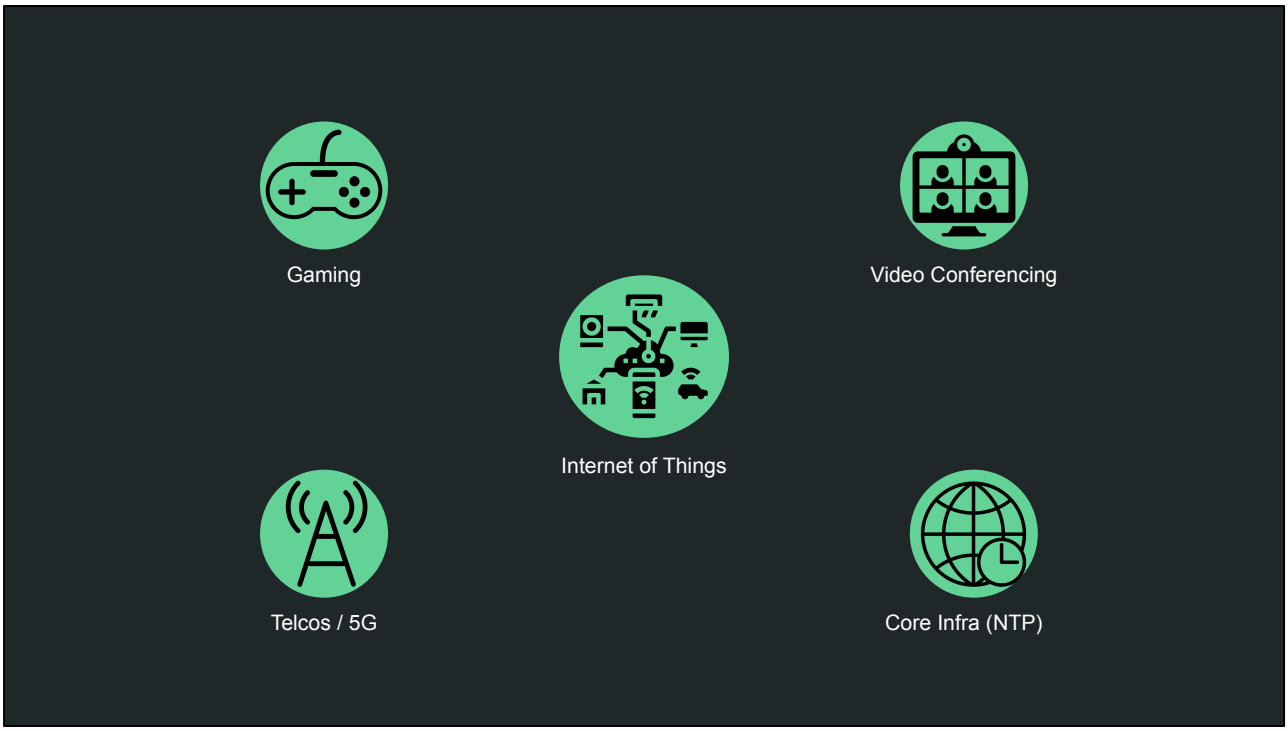


See, there are a ton of IoT protocols designed specifically for the needs of connecting physical devices to each other and to the internet. Some are optimized for power, some are optimized for bandwidth. Others implement standardized industrial or consumer control planes. Some are built on the internet protocol, like UDP & TCP, while some can't use IP!

As a startup we're interested in implementing many of these protocols in our cloud solution.



The challenge my startup faces in bringing IoT protocols to production is similar to DNS in that these protocols are less common than HTTP, a lot less in fact, and therefore aren't supported easily by many of the projects, services & clouds we might want to leverage. Therefore we were faced with the challenge of implementing load balancing, routing, etcetera from scratch, all by ourselves. That means we can't leverage the wealth of open source from this community or take advantage great solution providers.

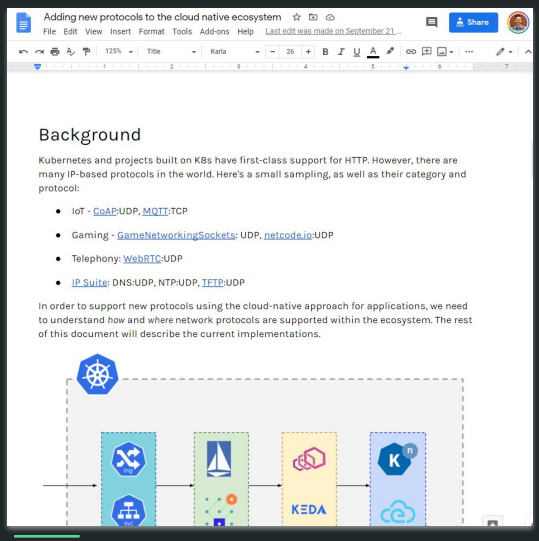


But I did say before that the challenge of implementing protocols isn't specific to IoT. I wanted to see if other people in the community are also implementing new protocols and see what I can learn from them. So I started talking to other folks, like game servers devs working on Project Agones, WebRTC maintainers from Pion, telco folks from the Network Service Mesh project and others. Turns out, they're all asking a similar fundamental question.

How can we implement
any protocol
in a cloud native way?

How can we implement any protocol in a cloud native way? This is clearly a community challenge, something bigger than my startup or IoT.

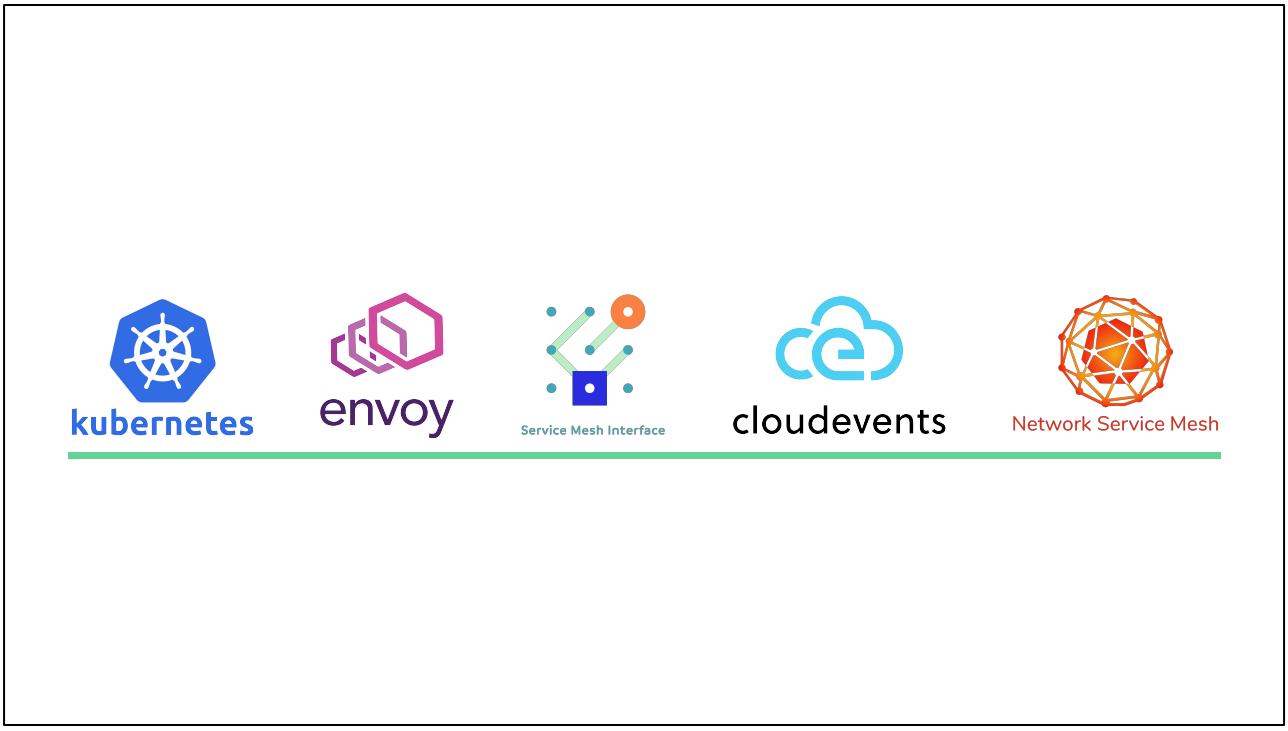
bit.ly/beyondhttp



I'm a PM so I did what we do best - start a doc. You can go to it now at bitly slash beyond http. It's a living doc that surveys a growing list of cloud native projects that might be used as part of a solution that implements a networking protocol. It tries to identify which projects are specifically focusing on supporting non-HTTP protocols and suggestions on where they can add additional hooks that we implementers might need.

Feedback and contributions greatly welcome.

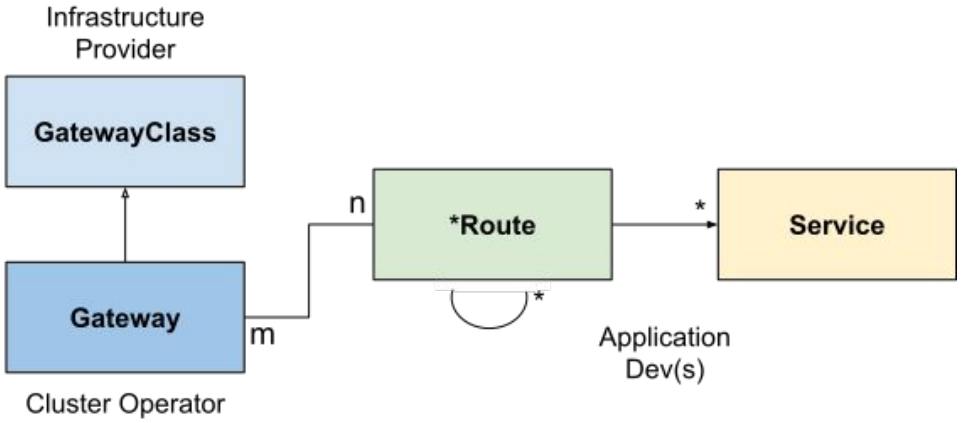
I started the doc over a year ago and since that time contributors and maintainers have helped shape it. At the same time, projects across the cloud naitve landscape have added upstreamed features that make it easier to implement new protocols. This progress is both exciting and encouraging.



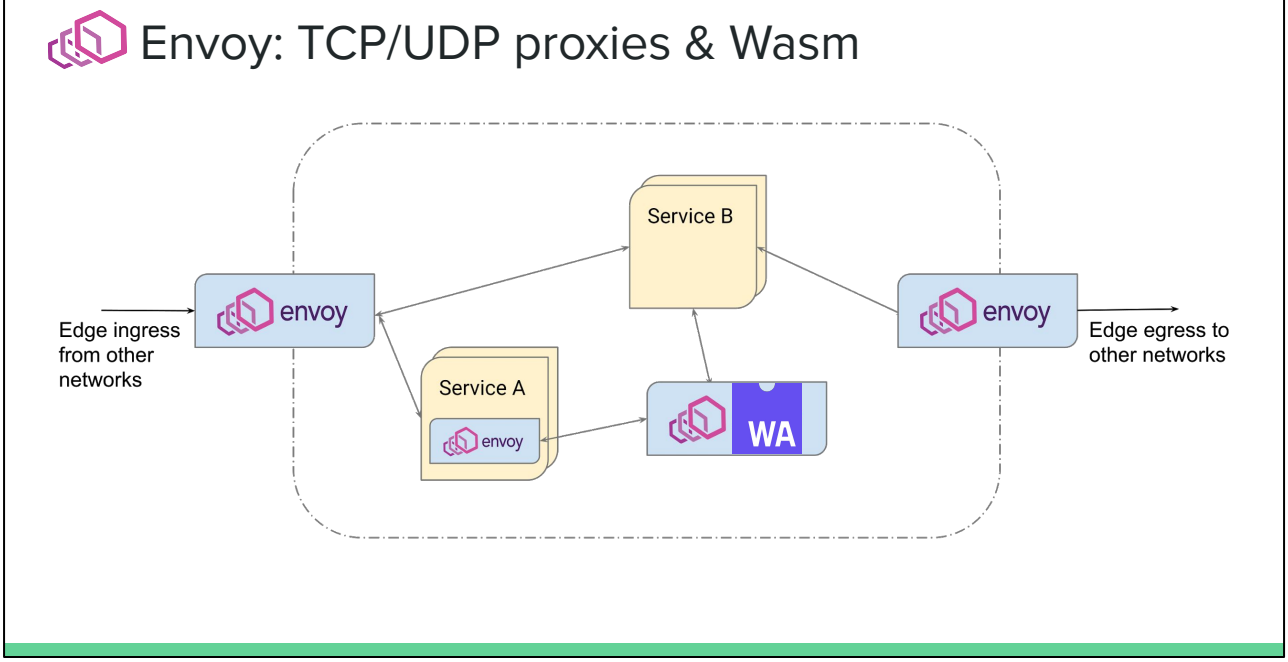
There's more projects than I have time to discuss in this session but I wanted to highlight a few I think will be used by the most amount of folks. Kubernetes, Envoy, Service Mesh Interface, Cloudevents and Network Service Mesh.

Let's start with Kubernetes

K8s: Gateway, aka Ingress V2

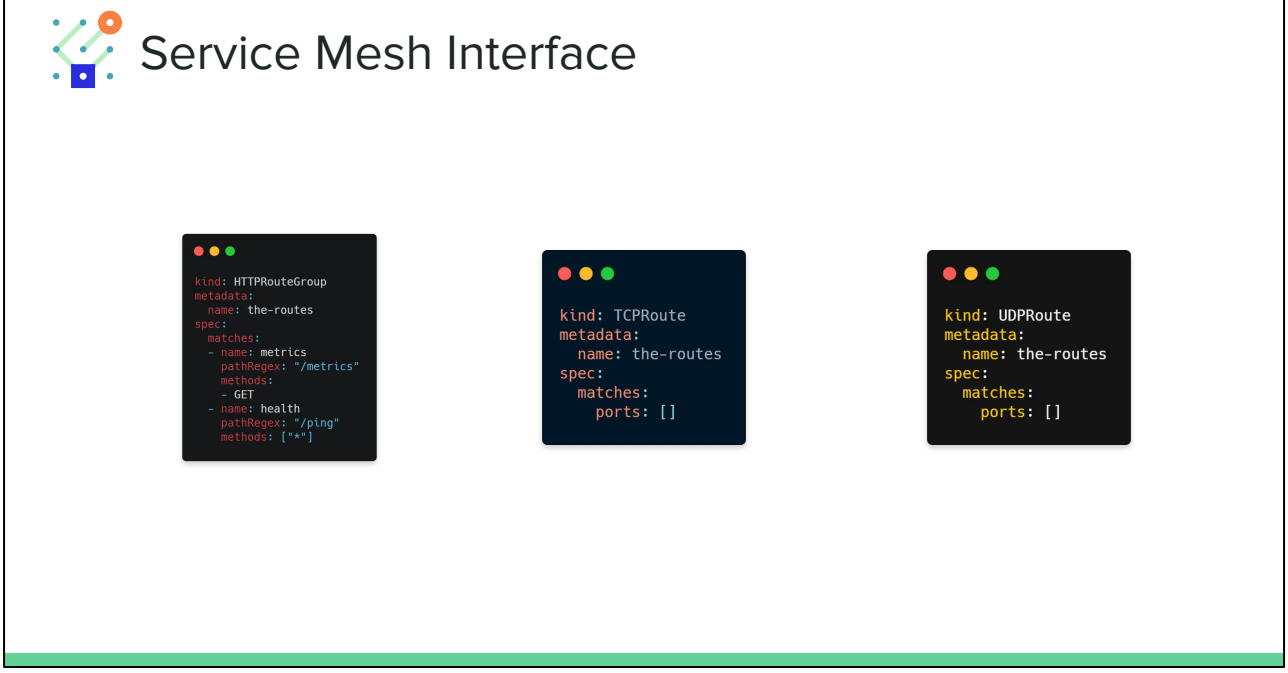


Kubernetes has a concept of Ingress, which is an object that exposes networking traffic from outside the cluster to services within. The current stable Ingress implementation supports HTTP, go figure. Within the Networking SIG, a group has been hard at work developing the new Gateway API to replace Ingress. Like Ingress, a Gateway routes traffic to a service but supports more protocols like UDP and TCP (among other things), with a goal to enable protocol implementers to create custom gateways in the future. They already have demos you can try and it may be in an Alpha release by the time you watch this recording.



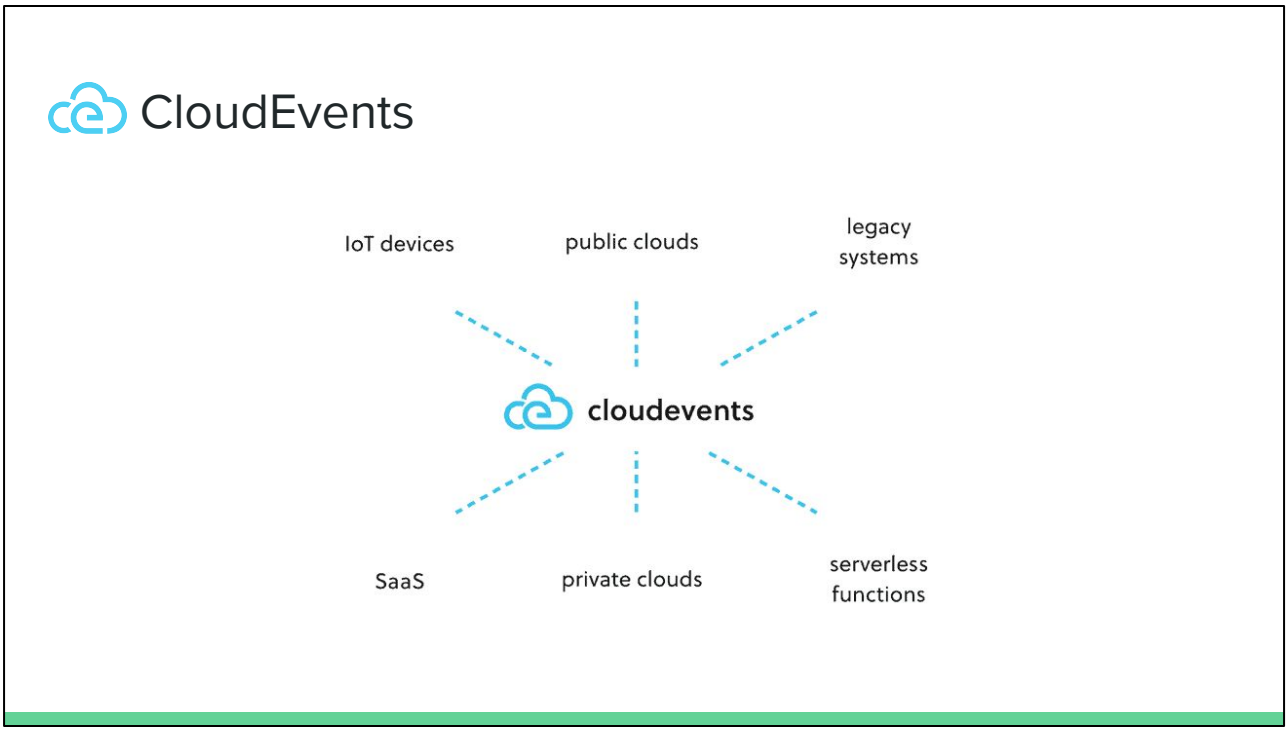
Envoy, as a popular networking proxy, by its nature needs to support a protocol in order to act as proxy for said protocol. Envoy has always had native support for HTTP but due it's popularity began implementing support for protocols like Redis and Postgres, albeit in a one-off fashion. Over the last year or two, the team has added UDP and TCP listeners, which makes it possible for protocol implementers to use Envoy to proxy any protocol based on those layer 4 primitives, which covers a lot of potential protocols. Also, with the introduction of WebAssembly support, Envoy has now made it even easier for more people to implement custom protocols in the language of their choice.

One more thing - since Envoy is used as a basis for other projects like Service Meshes and Observability tools, it now becomes easier for projects like Istio and Prometheus to support custom protocols in the future.



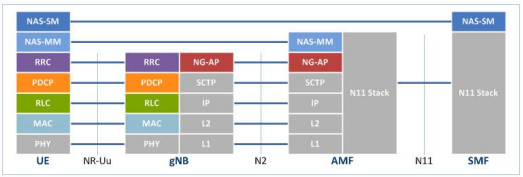
Next I'd like to highlight Service Mesh Interface, or S.M.I., which is developing a standard for service meshes on Kubernetes. One aspect of a service mesh is how it manages traffic between services within a cluster. SMI defines a concept of "Traffic Specs", a list of subspecs that define how traffic flows between the mesh. There are already sub specs for HTTP, TCP & UDP. The intent is for protocol implementers to define their own Traffic Specs and eventually contribute them upstream to SMI where it make sense.

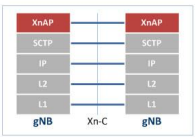
You may notice some parallels between the Gateway API in Kubernetes and the Traffic Spec here in SMI. Gateways are about traffic coming into a cluster while traffic specs are internal traffic within, so there's a potential for the two projects to work well together. I'm personally excited to see who these two efforts may collaborate in the future.

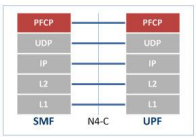


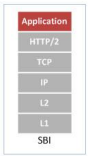
CloudEvents is a specification for describing event data in common formats to provide interoperability across services, platforms and systems. It defines how to serialize events in different formats like JSON and protocols - which of course includes HTTP. There's already subspecs for protocols like Kafka, MQTT & NATS and clear documentation on how to define custom slash proprietary protocols. CloudEvents is already being used today in projects like Knative and I believe will become more commonplace among event-oriented microservices, so having a way to define custom protocol bindings will make it easier for services leverging non-HTTP protocols to interoperate.

Network Service Mesh









AMF	: Access and Mobility Management Function	PDCP	: Packet Data Convergence Protocol	TLS	: Transport Layer Security
gNB	: Next generation NodeB	PFCP	: Packet Forwarding Control Function	UE	: User Equipment
MAC	: Medium Access Control	RLC	: Radio Link Control	UPF	: User Plane Function
NAS	: Non-Access Stratum	RRC	: Radio Resource Control	XnAP	: Xn Application Protocol
NAS-SM	: NAS – Session Management	SBI*	: Service Based Interface	Xn-C	: Xn Control plane
NAS-MM	: NAS – Mobility Management	SCTP	: Stream Control Transmission Protocol		
NG-AP	: Next Generation Application Protocol	SMF	: Session Management Function		

* SBI: Namf, Nsmf, Nudm, Nnrf, Nnssf, Nausf, Nnef, Npcf, Nudr, Nsmsf, Nudr, etc.

The last project I'm going to touch on is Network Service Mesh, not to be confused with the *other* kind of Service Mesh or the Service Mesh Interface. NSM is here to extend the existing kubernetes networking model but address networking use cases that need to go deeper in the stack, like L3 or L2 layers or even Ethernet frames themselves. Inspired by Istio, Network Service Mesh maps the concept of a service mesh to L2/L3 payloads, hence the name. Telcos, and the CNCF Telecom User Group, are big fans of NSM and one reason is how it enables them to implement cellular-specific protocols in a cloud native way. 5G is propelling the Telecom industry into fully embracing what they call "Cloud Network Functions." Many of the functions have been classically implemented in very expensive hardware found in cell towers that take a long time to physically upgrade. With Cloud Native functions, Telecom operators hope to build these capabilities as services that can be more easily be upgraded over time. Lot's of the functions are specific niche protocols that are part of the operation of 5G networks and NSM provides the means to configure them easily on top of K8s.

I'm only scratching the surface on what NSM enables but fun fact - because Service Meshes like Istio operate up the stack at L7/L4 and NSM is L3 and below, you can use a Service Mesh with NSM - and there's a bunch of advantage when combining the two concepts. I don't have time to dig into that now but the NSM crew have some examples.

Attribution:
https://www.netmanias.com/en/post/oneshot/14103/5g/5g-protocol-stack-user-plane-c

ontrol-plane

With that I want to conclude with what you can do to participate. If you're trying to create something with protocols other than HTTP, share your story. Documenting use cases, both real-world and aspirational, has been the single most effective tool to help improve protocol support. Create issues, hop on the CNCF or Kubernetes slack or just hit me up on Twitter. If you're a project maintainer, try to identify where your projects are currently assuming all traffic is HTTP and look for places for extensibility. And, ask your users what they think!

Attribution
https://unsplash.com/photos/ujx_KIIujRg

Thanks!

Jonathan Beri ◇ bit.ly/beyondhttp ◇ @beriberikix

And that's all I got! Thank you everyone. Here's that the link to doc one more time, and you can again reach me at @beriberikix. Cheers.