Edge 2.0 Core Principles

By Rajesh Narayanan, Michael Wiley

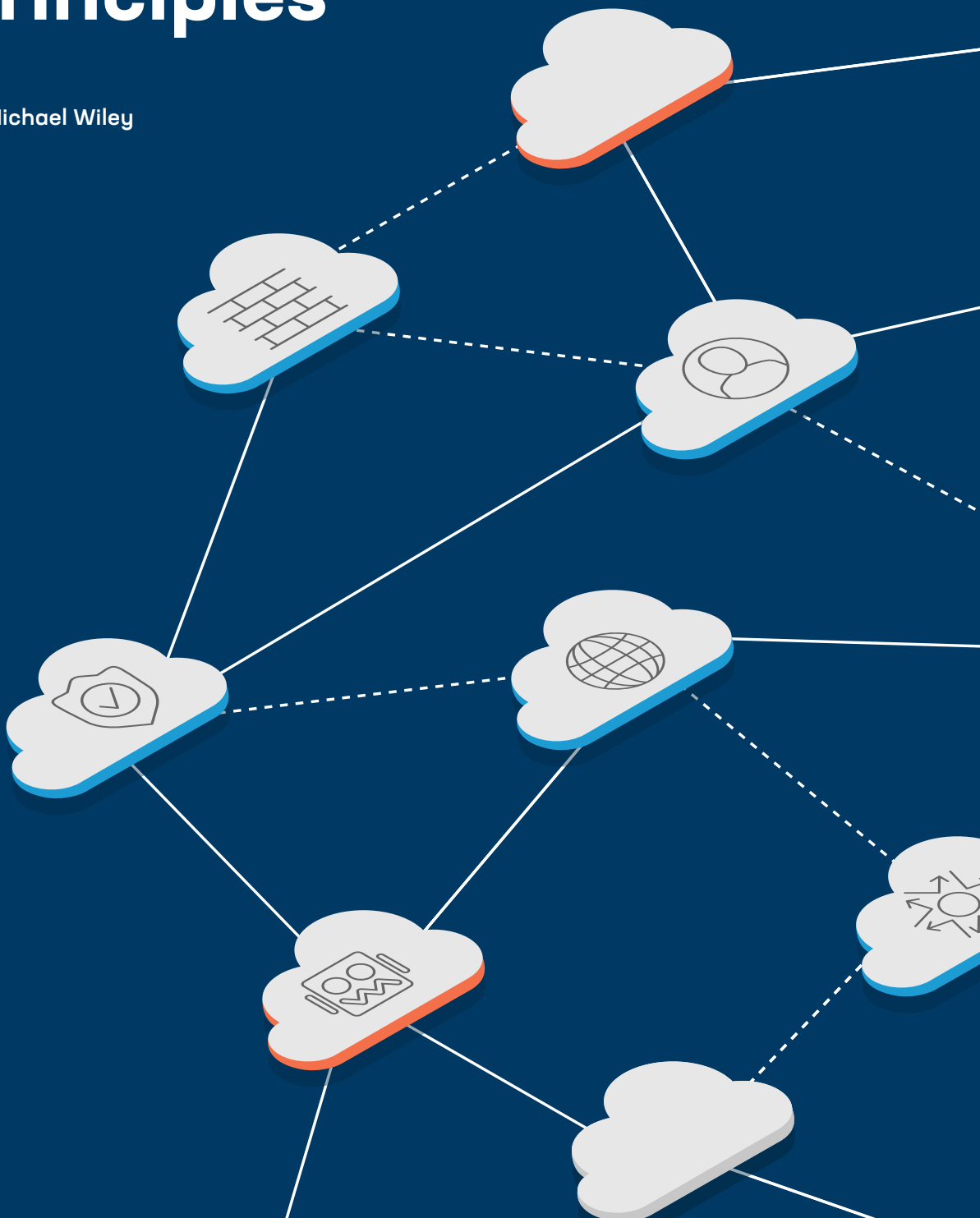

Table of Contents

Introduction

EDGE 2.0 IS ABOUT THE EXPERIENCE, NOT THE INFRASTRUCTURE OR ITS LOCATION

The definition of the edge has always been intertwined with the evolution of the data center architecture. The last decade has witnessed a rapid transformation in enterprise infrastructure architecture, expanding from on-premises data centers to today's distributed cloud architectures. We recognize Edge 2.0 as a technology shift that will enable applications to leverage a distributed cloud architecture.

Every person, organization, or entity has a different interpretation of the edge. Some might consider the edge to be a cell tower, others might say their mobile device is the edge. For cloud service providers (CSPs), the edge is a managed infrastructure footprint which seamlessly integrates into their back end. For military applications, the edge is the theater of battle—be it the land, sea, or air. Every time we read about the edge the interpretation is generally summarized as compute, networking, and storage capabilities of the infrastructure and/or its location.

But we also see Edge 2.0 as the experience it delivers to all the stakeholders in the ecosystem, and not just the infrastructure asset or its location.

This document describes what the functionality of Edge 2.0 should be, independent of its physical or virtual infrastructure, or where it is located or instantiated. The intent is not to explain how to build a better distributed application, but to illuminate the capabilities that must be in Edge 2.0 to support the creation of the most optimum distributed applications that suits a particular requirement.

WHAT IS EDGE 2.0?

From the point of view of an entity that resides on this distributed cloud, the edge is wherever the entity is currently located. Thus, we propose a definition of Edge 2.0 that is experience-centric, not tied only to the location of the infrastructure, type of infrastructure, or the controlling entity.

The focus of Edge 2.0 is towards enhancing the application-centric, operational-centric, and developer-centric experiences. Edge 2.0 must address the meta-properties (like SLAs and SLOs) of the application by adapting to the shifting needs of the application. Edge 2.0 must be simple to operate and unburden operations teams from creating new automation for every distributed application. Edge 2.0 must reduce the friction faced by developers when they develop and deploy distributed applications at scale, by seamlessly supporting security, governance, and service-level objectives.

As an example, let us take a banking application. The goal of Edge 2.0 is not to improve the business logic of the transaction. It is about making it more secure, faster, private, usable across all geographies (as needed), and scalable up or down as needed to support the business goals.

CONCEPTS AND ASSERTIONS

The following are the key concepts and assertions of this paper:

- Experience-Centric Edge: Establishes a basis for Edge 2.0 around the experience it delivers, instead of an asset or its locations.
- Design Considerations: Specifies key design considerations for a successful implementation of the Edge 2.0 architecture.
- Heterogeneous Infrastructure: Highlights that designing for a distributed cloud means considering decentralized control of the infrastructure.
- Telemetry as a Foundation: Emphasizes telemetry as a fundamental building block that must be supported across all layers of the infrastructure. Without telemetry a data-first strategy becomes diluted.
- Inter-Cluster Challenges: Underscores a fundamental challenge with Edge 2.0 as being inter-cluster instead of intra-cluster.
- Adaptive Interfaces: Introduces the adaptive interfaces, offering a distinct comparison with declarative and imperative interfaces.
- Edge 2.0 Application Platform (EAP): Introduces the EAP as a framework to enable Edge 2.0 to adapt to the needs of the applications.

UNADDRESSED TOPIC(S)

There are several topics we have not yet addressed in this paper:

- Application Data Distribution: There are many subtopics here like content delivery networks (CDNs), storage distribution, data-distribution, caching, etc. There are also emerging technologies like Conflict-free Replicated Datatype (CRDT). An Edge 2.0 framework should include support for application data distribution in its scope.
- Functions Distribution: The advancement of edge compute can be thought about as core cloud functions and logic, shifting closer to the location where the event is generated, or the data resides. Due to the significant amount of compute becoming available at the edge we should now solve problems similar to those typically solved in legacy cloud architectures if we wish to take advantage of that compute. For example, overlay networking, middleware workloads, and other types of workloads which are intertwined and are more complex than the naïve use cases we have seen in early edge –for instance storlets, which are stateless simple flows to be run next to the data.
- There are likely other areas not discussed. The goal of the framework is to be extensible such that the responsibilities can be added as needed.

Edge Evolution

Figure 1 best depicts the co-evolution of the Edge and datacenter architectures.Edge 1.0 and 1.5 were based on the notion of an origin site and moving the data and services from the origin to the consumer. Edge 1.0 was deployment of internet infrastructure primarily focused on optimizing bandwidth usage with distributed content caching, a.k.a content distribution networks. CDNs are a fundamental design pattern since the aggregate content to transfer will always be more than the available bandwidth.

EVERY ENTITY CONSIDERS ITSELF TO BE AT THE EDGE FROM ITS POINT OF VIEW

As CPU and memory costs raced downwards, compute farms appeared along with CDN nodes. Applications were consumed as services through service-oriented architectures (SOA). Hence the reference to Edge 1.5 as Service Distribution Network.

A common characteristic of Edge 1.0 and Edge 1.5 is the idea of an origin site. Before the growth of mobile, people, devices, and things were primarily downloading content or acting as consumers of the service. The site originating the service was still different from that of the consumer.

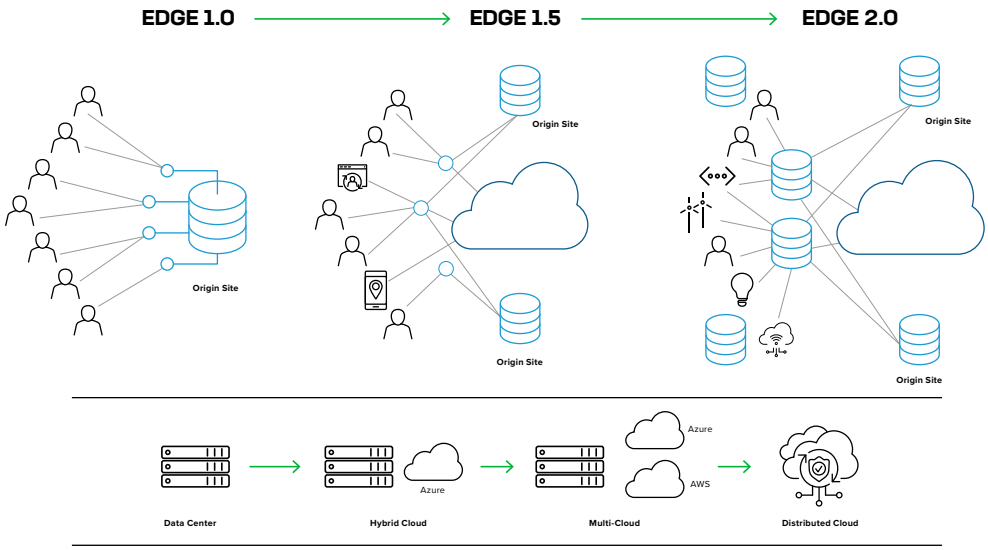

Figure 1: Edge and infrastructure co-evolution

In Edge 2.0, however, any entity can act as an origin site or as a consumer. The traffic flow has changed. Humans, phones, and devices are actively producing data as they upload content or generate periodic data. Apps are becoming consumers as they depend on other apps. All entities can act as producers or consumers of a service—APIs, humans, IoT devices, or web, back-end, and headless applications. From its own viewpoint, every entity on the infrastructure thinks it's at the edge.

The distributed cloud is the latest stage in the evolution of the datacenter. Compute has become truly ubiquitous, from mobile devices to being embedded in everyday objects that are connected to the network. Coevolved with it are software architectures to enable distributed and scalable applications.

EDGE CONSEQUENCES: INCREASED COMPLEXITY

The abundance and hyper-distribution of compute and storage everywhere has created an opportunity for rapid digital transformation of the enterprise. But this rapid evolution has its consequences.

Most enterprises are typically comprised of heterogeneous infrastructure, with non-uniform environments. Effectively scaling such environments demands complex orchestration and automation from the deployed systems. Rapid changes to the application's needs, such as changes in CPU and bandwidth requirements, increase the complexity of operations across a distributed cloud. This adds stress to the operations teams, who may not be well-equipped to efficiently respond to the changing needs of the application or the end customer.

The consequences of these issues are operational complexity, which neutralizes any potential gains for an enterprise going through digital transformation.

DIGITAL TRANSFORMATION GAINS ARE NEUTRALIZED IF THE TECHNOLOGY IS OPERATIONALLY COMPLEX

Some of the intertwined issues and artifacts that result from complexity are highlighted next.

Security models need to constantly keep up

Hybrid clouds result in increased surface area that can be compromised due to a variety of attack vectors. Different use cases create multiple security challenges, and as the infrastructure evolves, we are constantly chasing the puck.

Thus, the problem we anticipate is: will only the technology recommendations change often, or will the design patterns to implement security change as well?

Here are some of the issues with existing approaches:

- Software Defined Perimeter (SDP) does not scale easily, as popular solutions are agent based, which does not lend to simple operational deployments.
- Implementing Zero Trust Network Access (ZTNA) is often impractical, due to ZTNA solutions requiring a constellation of deployed production services like traffic inspection, centralized log management, global PKI and identity management, and more.
- Secure Access Service Edge (SASE) combines network-as-a-service and security-as-a-service, an acronym soup of several technologies non-trivial to implement. Additionally, the focus of SASE tends to be software-defined wide-area network (SD-WAN) centric, addressing a small segment of the enterprise networking edge use cases.
- Disparity between different public cloud providers' infrastructure and already complex security models (for example IAM) often lead teams to a patchwork hardening of providers and cumbersome multi-cloud configurations.

SECURITY IS A CHASING-THE-PUCK PROBLEM

Automation Challenges

One of the primary challenges with the hyper-distribution of low-power and low-cost compute available at the edge is the ability to orchestrate and schedule the infrastructure in a uniform manner. Ops teams struggle with scaling and supporting applications that leverage the distributed cloud—as these applications typically include heterogeneous technologies with different administration requirements.

While automation platforms like Terraform provide a sophisticated means to customize the automation, ops teams still need to maintain scripts for multiple permutations: for example, five different flavors of compute, four different types of firewalls, and three types of load balancers. The human cost of having to manage and maintain the automation scripts does not scale.

This leads to the infrastructure customer's continued interaction with ops teams via ticket driven systems, which are a barrier towards automating existing workflows. Service desk gets overwhelmed with tickets, needing to prioritize security and stability over agility.

API Sprawl

Microservices architectures have already become the de-facto method of building new applications with the evolution towards multi-cloud. APIs are published and can be instantiated whenever and wherever needed, leading to API sprawl. The discovery, connectivity, and management of these APIs in an autonomous manner becomes a big challenge.

A paradigm shift is needed in addressing API sprawl. Our internal models demonstrate that even moderate assumptions of parameters, like the number of global API developers, the number of APIs developed per dev per year, and API shelf-life, result in many 100s of millions (if not billions) of APIs being simultaneously active by 2030 (Figure 2). The 2021 API Sprawl report provides a comprehensive view of this emerging topic.

Poor End-to-End System Visibility

Increased complexity requires enterprises to enable granular and meaningful end-to-end visibility of the end-to-end system. In distributed cloud environments, applications transcend multiple heterogeneous infrastructure stacks managed by separate entities.

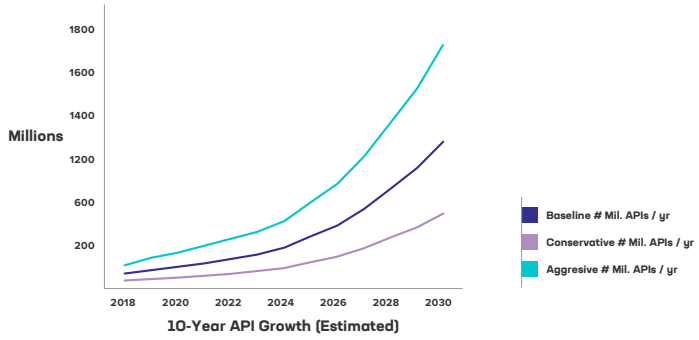

Figure 2: 10-Year Estimated API Growth

Moreover, none of the operators today are incentivized to expose native telemetry of their infrastructure to enterprise customers. Enterprises are essentially running blind when deploying applications in a distributed cloud and need to resort to multiple observability tools. To fill these visibility gaps, the tools typically are from different vendor companies working at different points in the infrastructure or the application stacks.

Non-standard infrastructure metrics add to the woes within an enterprise. Typically, the metrics are not unified due to operational silos and other factors like non-uniform infrastructure in different infrastructure environments. For example, metrics across public cloud providers are different, and on-prem data center technologies also do not follow any standard metrics.

BUSINESS OUTCOME: DIMINISHED EXPERIENCE

MANY ENTERPRISES RUN BLIND DUE TO A LACK OF AN "OBSERVABILITY PRACTICE"

Revenue-generating workloads and mission-critical systems typically utilize most of the operational resources and budget available in an enterprise. Meanwhile smaller projects, though lower in complexity, tend to make up the volume of the enterprise's applications. Figure 3 depicts this as a long-tail distribution of the number of projects that an enterprise is likely to have in comparison with its operational complexity.

While smaller applications may be lower in operational complexity, the operationalizing processes and workflows are still in many cases manual, and it can take weeks for service tickets to successfully transit multiple operational teams. For instance, deploying an application that requires dedicated network resources with granular security policies may first require the global security teams to work out the security policy approvals. Then the service ticket may go to the networking team for planning the subnet and route configurations that need to happen. Finally, the validation of firewall rules may be required from yet another operational team that is responsible for firewall management.

Now let us imagine the above complexity needs to be addressed for each application deployed on a distributed cloud where the enterprise does not own any part of the underlying infrastructure. The sheer volume of small projects or applications that need to be onboarded and supported makes it unrealistic for operations teams to support every project, creating a prioritization issue and self-service opportunity.

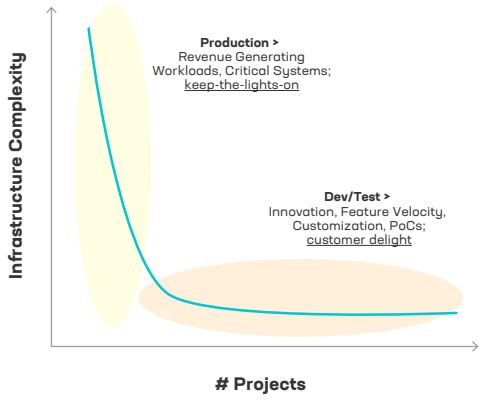

Figure 3: Distribution of project size vs its complexity of deployment

This prioritization issue is especially noticeable as infrastructure teams' investments are focused on supporting critical and revenue-generating systems, leaving little time or budget for net new projects that involve their own lifecycle of development, testing, and staging before production. This results in diminished capabilities and investment towards feature velocity, customization, and the innovation that supports new products, which culminates in stagnation of the business and its offerings.

Edge Solution Space

THE INFRASTRUCTURE TEAMS ARE BURIED IN COMPLEXITY

The evolution of the edge ecosystem greatly expands the solution space by offering an opportunity to address these challenges with an application platform.

Figure 4 shows the Edge 2.0 solution space. Here we state everything that comes under a distributed cloud architecture is the totality of the solution space.

Thus, within the context of the solution space, Edge 2.0 must deliver the experience desired by all of the following:

- All entities that reside within the solution scope—apps, humans, and devices that act as consumers, or web backend and headless applications that make up the services.
- The SRE teams and application owners of these apps, while preserving the security and regulatory guard rails expected of the infrastructure.

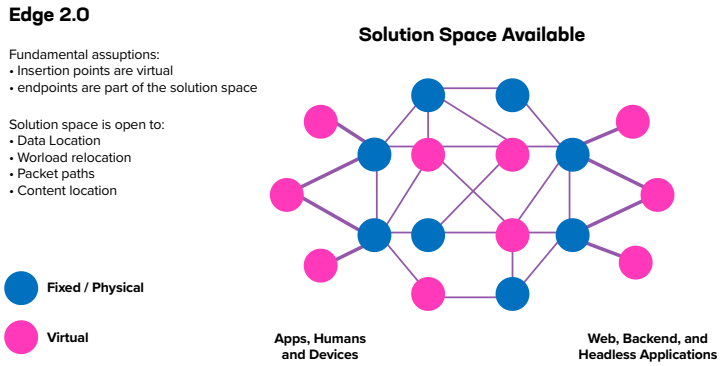

Figure 4: Edge 2.0 solution space

Edge 2.0 will be operationally simple, secure, compliant, and deliver a high-quality experience to all the ecosystem entities that participate in it.

Edge 2.0 Design Considerations

SECURE BY DEFAULT

A distributed cloud amplifies this security-at-scale problem as the number of endpoints increases exponentially. With such a massive scale, the complexity of implementing a solution also scales. The security posture should be that the application assumes the environment it is currently deployed in is already compromised. Similarly, the infrastructure shall not trust any application that runs in it.

The basis of these ideas is captured in the concepts of the Zero Trust mindset, and the BeyondCorp[1] exemplar demonstrates these concepts for the use case of application access. Going forward, Edge 2.0 extends this model, based on the following five core principles:

(a) Identity Is Foundational: In Edge 2.0 identity runs deep, with each entity instance having its own globally unique identity, but also its place in the org hierarchy along with its privilege level. Identity should be a key consideration not only for north-south traffic, but also internally for east-west access.

(b) Least Privilege: The actions allowed by an actor should be constrained to only those that the actor requires to perform its role in the system. An example is limiting the subset of workloads that are allowed to communicate with the database server based on the design specification.

(c) Continuous Authentication: Any attestation that a system actor makes must have a means of verification and should be explicitly verified whenever such an attestation occurs. Authentication should not be solely explicit through shared secrets or chain of trust, but also should factor in implicit patterns of behavior and contextual meta data.

ALWAYS ASSUME INFRASTRUCTURE IS ALREADY COMPROMISED. THE CHALLENGE THEN IS TO SAFELY DEPLOY AN APPLICATION ON IT.

(d) Constant Monitoring and Assessment: The actions of all actors in the system must be monitored, reinforcing the key role of data collection and warehousing technologies in an Edge 2.0 architecture. In addition, these activities must be continuously assessed to detect and mitigate attempts to perform forbidden or dangerous actions. The assessment must be near real-time, at scale, implying liberal use of automation and machine learning techniques.

(e) Assume Breach: Given the resources available to today's sophisticated adversaries, one must assume that any system has been compromised in some manner. As a result, the entirely deterministic policies rooted in workload and user identities, enforcing privilege-based access represented by a and b, are often insufficient to prevent all attacks. Therefore, a fully mature Edge 2.0 system must also be able to make real-time risk/reward assessments based on continuous monitoring and assessment.

PROVIDES NATIVE OBSERVABILITY

Edge 2.0 must integrate telemetry as a first-class citizen deep within the infrastructure stack, provide a simple and scalable means to collect cross-layer telemetry within the infrastructure, and surface it to a common platform. This helps observability teams to surface interrogation of the "infrastructure state" as needed. This lets application teams focus on the business logic without explicitly instrumenting their application stacks.

Figure 5: Evolution of the telemetry collector

The current state of observability technology has largely been vendor specific and proprietary. This has led to many vendor-specific agents collecting similar signals that jostle for expensive memory and CPU.

The next logical step is the use of a vendor-agnostic telemetry collector that enables infrastructure and traffic data to be streamed to a centralized data platform.

Many product teams struggle to justify the investment in instrumentation, as the effort needs to map to a revenue opportunity. Infrastructure should be instrumented like any other business function or process, because the team needs to understand its behavior to optimize it for the business's objectives. Thus, the debate should be more on what should be prioritized to instrument, as opposed to its need.

INSTRUMENTATION MUST SHIFT LEFT

To achieve granular and more accurate measurements of the application behavior we anticipate the instrumentation will "shift left" by invoking it at the time of code—Figure 5.

SUPPORTS ADAPTIVE INTERFACES

In keeping with distributed cloud, peeling a couple of layers down each cloud within its scope (local or global) has its own manager, admin, orchestrator, and more. Each behaves independently, making its own decisions, but providing interfaces for other entities to use as needed.

Thus, the notion of a distributed cloud, in essence, is decentralized administration and control. One cannot get away from this fact, and it is important for us to recognize to better realize the nuances of adaptive vs. declarative and imperative interfaces.

To effectively utilize the Edge 2.0 infrastructure, imperative and declarative interfaces aren't enough as they still rely on hand-crafted artifacts that are essentially static constructs that do not adapt fast enough to rapidly changing conditions.

PROCLAMATION: WE DECLARE IT IS IMPERATIVE FOR EDGE 2.0 TO SUPPORT ADAPTIVE INTERFACES

Outcome based is where we need to go, and the systems need to be smart enough to adjust policies across the infrastructure (end-to-end) to meet those outcomes. We call these adaptive interfaces.

To be clear—imperative, declarative, and adaptive interfaces are not mutually exclusive:

Imperative: Gets very prescriptive and defines in detail a series of actions to get to the desired state. Configuration of a router, for instance, is an imperative action. In the above scenario, the prescriptive actions will be precise—which datacenter, how much CPU/memory, bandwidth required, specific routes, and more. The input quality of the data model is very high and thus requires a deeper knowledge and expertise about the infrastructure. There is an expectation of knowing both the model and structure of the infrastructure.

Declarative: The nuance of declarative is that it focuses describing the desired state, as opposed to actions that achieve the desired state. The quality of input is lower, though it still requires the application to know at least the structure of the underlying infrastructure.

Adaptive: Stands separate from imperative or declarative. An adaptive interface focuses on the desired objective or goal, rather than the state. The adaptive interface's goal is to capture the objective of the stakeholder who wants to deploy the application rather than focus on a pre-knowledge of the underlying system. Adaptive is different as it has no notion of what the capabilities of the underlying infrastructure are. There are no constraints on how to "get the job done" and, hence, it stands on its own.

With adaptive, the quality of the input is low and approaches natural language. Application owners can send a request to tell the infrastructure controller what outcome they expect, instead of needing to declare the capability required or specifically configure a resource.

SOLVES THE INTER-CLUSTER PROBLEM

A distributed cloud, by definition, accommodates all types of application architectures: monolithic, microservices, or serverless. Irrespective of the architecture, applications use APIs to offer the services.

The problems of API discovery, connectivity, and network security are largely addressed by using a service mesh, but a service mesh only solves this within the cluster (intra-cluster).

Edge 2.0 applications on the other hand leverage APIs across a hyper-distributed cloud infrastructure, in essence a multi-cluster environment. New applications are written with APIs crossing organizational and geographic boundaries. Some of the APIs might be private or partner APIs behind multiple layers of firewalls without an explicit route between them, even if they are discoverable. This heterogeneous cluster (inter-cluster) problem is without an elegant, lightweight, scalable, and secure solution that is practical.

Table 1: Comparison: Imperative and Declarative vs. Adaptive

Scenario/Properties	Imperative	Declarative	Adaptive
Definition	The imperative interface defines the detail control flow based on the specific capabilities of the underlying resource.	The declarative interface defines the logic but not the control flow. From a programmatic parlance it is the pseudocode.	The adaptive interface expresses the desired state as a requirement without making any assumptions of the capabilities of the underlying resources. An adaptive interface is owned by the infrastructure and enables the infrastructure to dynamically respond to the changing needs of the application.
Scenario 1: Package needs to go from SFO to NYC	Jane tells Mark: (a) Print the shipping label, (b) Go to the UPS store, (c) Pick the 72-hour delivery, (d) Pay for it and ship it Mark: Has to follow a very specific set of steps	Jane asks Mark: (a) Please courier this package to this address, (b) Package must arrive within 72 hours. Mark: Can pick any courier company (UPS, FedEx, DHL etc).	Jane expresses to Mark: (a) This package must arrive in NYC by Saturday. Mark: Can do whatever he likes. Potentially even taking the package himself, flying to NYC, and delivering it by hand.
Scenario 2: Load balancer example	The load balancer already exists. Task: needs to be configured with this policy. Here the task is specific to the load balancer make, model, version, etc.	A load balancer does not exist. Task: load balance the application with a given open policy. Task assumes that an orchestration or management layer exists and declares what needs to be done. Action: Ultimately task is converted to an imperative interface to configure a specific load balancer.	No assumptions about the underlying infrastructure. Please ensure the application scales as needed, with a max latency of 10 ms.
Ownership	Joint ownership: application and infrastructure	Joint ownership: application and infrastructure	Only the infrastructure
Quality of Input	Extremely high. Requires deep knowledge of the underlying scope. For instance, need to know which F5 load balancer, which software version, etc. A CLI or NMS would be an example of imperative interfaces.	High: Requires knowledge of what the infrastructure is capable of. For instance, in the above example there is pre-knowledge of the infrastructure supporting a load balancer.	Low: Requires no knowledge of the capabilities of the infrastructure. The input quality approaches natural language equivalent.
Skill Level (Persona)	Function-specific skills. For example, network admin, compute admin, storage admin. Every aspect of the infrastructure requires the user to be an expert in that area.	Application deployment skills. The user knows the type of function required to deploy the application. As in the above case, the application manager knows a load balancer is needed, and high-level policies on which the LB must be configured to support autoscaling.	Application engineer. Can just signal the infrastructure what the non-functional requirements of the application are. In this instance, how fast the application should respond to a customer request. Other factors like geographic location, cost, etc., can be added as needed.
Scope	The imperative interfaces have a highly localized scope. For instance, infrastructure, network, data center, rack-level, etc.	The declarative scope is larger. Typically, associated with an orchestration engine that talks to multiple imperative interfaces to achieve the outcome required.	Very large scope because the outcome of the interface can call multiple declarative or imperative interfaces. The execution is more complex requiring sophisticated implementations of imperative and declarative interfaces.

Example	`- name: update web servers`
	`hosts: webservers`
	`remote_user: root`
	`tasks:`
	`- name: ensure apache is at the latest version`
	`yum:`
	`name: httpd`
	`state: latest`
	`- name: write the apache config file`
	`template:`
	`src: /srv/httpd.j2`
	`dest: /etc/httpd.conf`
		`apache-server:`
		`version: "latest"`
			`application-latency:`
			`- lt: 10ms`
			
			`infrastructure-cost:`
			`- lt: $200`
			`- billing: monthly`

We covered APIs extensively in the 2021 API Sprawl report[2] and address within it many questions that might surface. Figure 6 shows the difference between inter-cluster and intra-cluster.

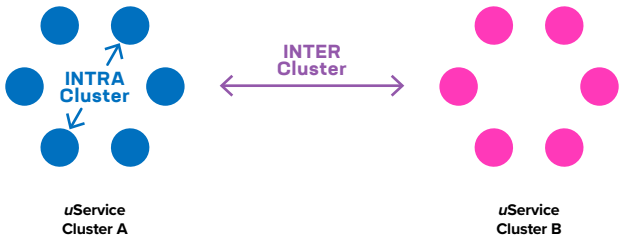

Figure 6: Intra-cluster vs inter-cluster

Intra-Cluster: In a monolithic architecture, there might be very few APIs exposed with internal communication between modules via other inter-process communication methods. Whereas a micro-service architecture is broken down into dozens, if not hundreds, of APIs.

Whatever the case, each application cluster is developed and managed in an opinionated manner. For example, in cases of microservice architecture a team might use any flavor of service mesh—Istio, Linkerd, Aspen Mesh, or others. Every approach has its own means to manage and control the APIs within its cluster, in other words, intra-cluster.

There are many solutions here, and the goal for Edge 2.0 is not to re-invent or force orgs or dev teams into adopting yet another new technology.

Inter-Cluster: As the number of services delivered via APIs increases, new applications are built using services already developed or adopted by different application teams within the enterprise, as there is no need to reinvent the wheel.

New applications are also being built through different combinations of public and private APIs. From an architecture standpoint, modern applications leverage services provided by other clusters. In a distributed cloud these clusters are deployed globally, hence, can be located on any real estate that can host the application or be part of the application cluster.

Scope of an Application Cluster

To reiterate, the scope of an application cluster does not limit itself just within an organization. Clusters can be across any two networks, applications, organizational silos, or even between two enterprises.

The scope spans the full gamut of all permutations and combinations, thus exponentially increasing the complexity of operations. Figure 7 shows the traffic permutations within the application deployment scope.

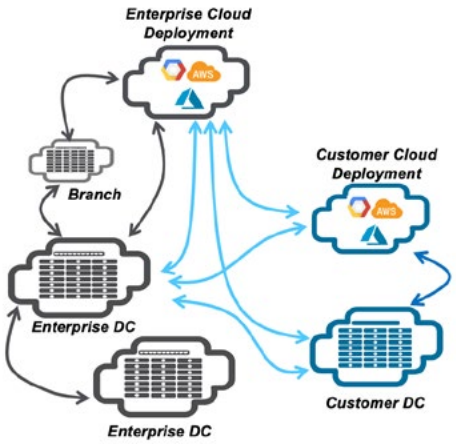

Figure 7: Traffic-flow permutations in modern enterprises

A typical enterprise will have different application architectures. There could be different flavors of service mesh, or a web 2.0 application based on service-oriented architecture, or a monolithic software deployment, depending on which team is developing and deploying it. APIs are thus scattered across the entire enterprise, be it private APIs or use of public APIs. This problem is not yet solved. API communication between multiple locations is critical and a problem with an elusive solution in enterprises of any size.

There are several solutions to manage APIs within a cluster (for example, ingress controller, API gateway, and more), while inter-cluster API communication is unsolved in a practical, scalable, and secure manner. Thus, we focus the scope of unified API control and management to only address inter-cluster issues.

DELIVERS AUTONOMY

An underappreciated value of the cloud is the autonomy it offers to the “cloud consumer.” Enterprises and entrepreneurs can instantiate their own infrastructures on-demand while experiencing a semblance of full-control.

The fundamental design principle an Edge 2.0 platform must follow is to deliver an autonomous experience to the cloud customer while implementing the right guard-rails. Since the entities can appear in any infrastructure (trusted or non-trusted) the guard-rails must be implemented in such a manner so as not to burden the BU or the DevOps team responsible for building the application.

Summarizing the tenets

The emerging challenge with Edge 2.0 will be that of discovery, identity, trust, and observability between various services.

Even in medium sized enterprises the number of APIs produced yearly would be large. How do teams discover these services within the enterprise? Or for that matter, how can they know

if the services are still valid and who owns it? Can they be sure this is a validated service with an established trust? The problem is compounded when teams want to consume services created by clusters residing outside the enterprise boundary, for example, SaaS providers or apps that run on edge devices completely out of the administrative control of enterprise infrastructure teams.

Edge 2.0 Application Platform

In consideration of the challenges presented in earlier sections, we need to view the entire distributed cloud as a platform. At an uber level we articulate the goal of the solution: to autonomously discover (without human intervention), scale, and secure distributed application across decentralized infrastructure.

Autonomously discover (without human intervention), scale, and secure distributed application across decentralized infrastructure. In essence we can describe the responsibility of the Edge 2.0 Application Platform (EAP) as follows:

- API discovery, identity, and trust
- Infrastructure scheduling and orchestration
- Secure application connectivity

AUTONOMOUSLY DISCOVER, SCALE, AND SECURE DISTRIBUTED APPS ACROSS DECENTRALIZED INFRASTRUCTURE

EAP SCOPE

The infrastructure is the totality of the solution space where infrastructure elements can continuously appear and disappear. The ownership of the infrastructure and its resource vs. the administration and control of those resource are two separate constructs. An infrastructure owner can assign a specific infrastructure or an instance of it to a different organization that manages and configures it, or the infra owner can take back control as required. The organization that manages and configures the elements can further assign it to individual projects or applications. This notion is not new; for example, cloud service providers offer hierarchical administrative control that can be used by an enterprise's IT teams to further offer Infrastructure-as-a-Service (IaaS) to internal business units, teams, etc. The main concern for Edge 2.0 is how to pull this off extensively in a hyper-distributed cloud, which is the ongoing and future state of the edge infrastructure.

This is where the EAP's scope comes into view. Figure 8 below shows the scope of EAP within the context of different types of interfaces. Each EAP orchestrates with its local scope using declarative and imperative interfaces. In this instance:

- Declarative / imperatives would be the AWS APIs, Azure APIs, etc.
- Adaptive interfaces would be net new that have to be defined on a case-by-case basis.

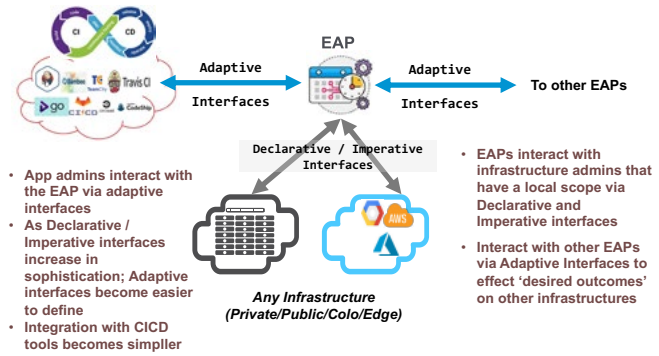

Figure 8: EAP scope mapped to interface types

To take advantage of the distributed cloud, the EAP should have the capability to leverage all the nodes and entities available through a distributed cloud. The vision is for the individual EAP to become equivalent to the autonomous system[3] concept in IP networks, but for the application layer.

Putting it together (Figure 9 below) we can now construct a high-level view of how multiple EAPs can be deployed across a decentralized cloud infrastructure interacting with each other in an autonomous manner.

Adaptive interfaces also make it easy for CI/CD and other application life-cycle apps to integrate with the distributed cloud. CI/CD platforms can directly interface with the EAP with simple adaptive interfaces stating only the desired outcome.

It is important to note that inter-EAP communication can also be achieved using adaptive interfaces.

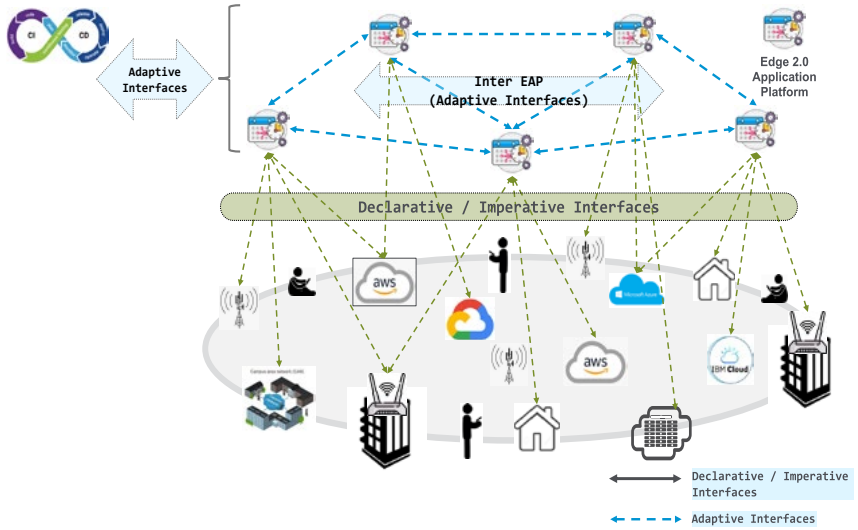

Figure 9: High-level topology of EAPs with local scope

THE EAP FRAMEWORK

The EAP framework, as shown in Figure 10, organizes the responsibilities in terms of capabilities of each EAP instance. The EAP's role is to interface with the underlay infrastructure controllers—be it Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS)—and to orchestrate and schedule appropriate resources as needed.

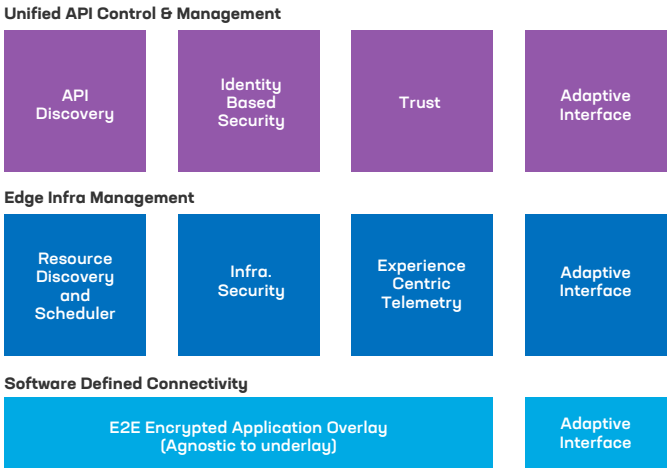

Figure 10: The Edge 2.0 Application Platform (EAP) framework

UNIFIED API CONTROL AND MANAGEMENT

The future will be an API driven economy, where APIs are more than just a software architecture approach simplified through service mesh. An API will become the primary means by which any entity participating in a distributed cloud provides its service.

As established, the global number of public and private APIs that will be available will be in the 100s of millions, if not billions, within the decade. APIs will reshape our economy and thus demand a closer look at what the EAP must implement to support this economy.

API Discovery

The impeding API sprawl demands that each API must be discoverable within and outside of the EAP. With distributed cloud, the APIs can be anywhere across multiple clusters.

The underlying assumption is the API discovery problem is truly an inter-cluster issue. An EAP's scope may span many application clusters that use different software approaches (microservices to monolithic), each implementing its own API gateway strategy. An EAP provides a common repository for all the APIs to be registered and discoverable within its scope, not just intra-cluster. This distinction is key to derive the limitations of existing API-gateway strategies, for example, as in service mesh.

The challenge for the EAP is to enable the discovery of an API, provide the right documentation on its usage, and manage the lifecycle of the API for consistency, accuracy, and versioning. The EAP implements a means to inform entities using its APIs on the current state of specific APIs being used. This could be simply by setting expiry dates or explicitly informing via some messaging system.

Identity-Driven Security

The security posture adopted is where an application assumes the infrastructure it's currently deployed in is already compromised.

The key pillar to implement this security posture is identity driven. Every API endpoint must have a globally unique identity. Security policies and controls are maintained separately in a central repository.

APIs must be marked public or private, triggering the underlying infrastructure security components to be automatically configured for the specific API.

All app-app conversations start with a deny-all policy. Just because an API is visible does not mean that another application can call it. This must be explicitly configured within the security policy of the application.

Trust: Reputation over Time

EVEN TRUST HAS A FINITE LIFE, IT MUST BE PERIODICALLY VERIFIED

The EAP must ensure all APIs within its scope can be trusted, and at the same time all external APIs being used by services within its scope can also be trusted.

"You can't prove trust, that's what makes it 'trust'" – From Traitors @ Netflix

Trust is essentially "reputation over time," and you must continuously revalidate that trust to ensure it has not dropped below an acceptable level. This has increasingly become a common approach in modeling and implementing trust in systems, requiring trust to be continuously assessed instead of statically asserted at initial execution.

The fluidity of trust over time may be a challenge to some enterprises that do not have the luxury of time to establish mutual reputation between their and third-party systems. Infrastructure and APIs alike can wreak havoc if the reputation is not closely watched.

Assuming a service within the EAP scope accesses an external API, the platform must implement a mechanism by which the calling service can be assured of the accuracy of the received response from the external API. Just because the API response looks valid, does not mean it can be trusted. Either the response could be inaccurate due to quality issues, or inaccuracies could have been explicitly inserted to make the business less competitive. Having this ability to assign a trust factor to each API, internal or external, is unique to the EAP construct.

One strategy to implement trust is to average it across a most recent “time period” instead of the full history of the service. This is like looking at “Top Reviews” vs “Most Recent” for a product in Amazon. The product may well have four stars, but if most of the negative ratings are in the past six months, this indicates a recent break in trust, while an influx of positive comments in the past six months would indicate a fix or rebuilding of trust.

The trust factor is not just specific to whether an API is a known source of false or misleading data or not. The reputation of an EAP will also depend on how well it manages and updates the APIs within its scope. Additionally, “trust” is also relative. Service A can trust Service C, but Service B may have a different experience with Service C.

EDGE INFRASTRUCTURE MANAGER

With a distributed cloud being the basis of Edge 2.0 infrastructure, it becomes imperative that the resources within the scope of an EAP be easy to discover, secured, and instrumented. The following can be read as a set of recommendations required of the edge infrastructure manager’s capabilities.

Discovery and Scheduling

Within an EAP, resources may get scheduled as needed. New resources may arrive or leave dynamically due to mobility. An EAP can also send or receive requests from other EAPs to discover and schedule resources as needed on its behalf.

Security

To reiterate the security posture: the infrastructure on which the application is deployed is already compromised. The EAP must thus ensure security of the application deployed within its scope.

Irrespective of the administrative level, the EAP framework must consider multiple faces of security, such as (but not limited to):

External Threats: For example, distributed denial-of-service (DDoS) attacks and advanced persistent threats (APT). These can be mitigated using existing best practices in security like DDoS prevention, firewalling, and more. The recommendation is that it is required for all traffic.

Man-in-the-Middle: All traffic must be encrypted and cannot assume the application layer will do the right thing. This ensures basic data confidentiality from any traffic snooping devices and protects the integrity of the data from manipulation during transmission.

Internal Threats: The scope of the infrastructure layer must be constrained and primarily directed to protect itself. The recommendation is to prevent a resource within the infrastructure from launching an internal attack on the infrastructure manager.

Experience Centric Telemetry

If Edge 2.0 is about the experience it delivers and not the asset or its location, it naturally follows suit that telemetry must also be experience-centric.

The question is, whose experience are we referring to? The experience is that of any entity that resides in or uses infrastructure to produce or consume a service: apps, devices, humans, back-end applications, databases, and more. The experiential viewpoint is thus that of the entity. A service that an entity produces or consumes is directly related to the compute, storage, and networking resources allocated to it.

But one must not just stop at measuring the experience; there must be a means to remediate it as well.

Any entity consuming or providing a service can determine whether it is getting the experience it requested. However, in distributed cloud environments, it may be near impossible to explain what happened in the infrastructure layer that led to the poor experience.

High-level metrics like CPU utilization, memory, bandwidth, and latency measurements are not sufficient to determine why an application is not getting the requested[3] experience. Metrics need to be time granular and be collected deep within the application stack and whenever available from different layers of the infrastructure stack.

A robust, scalable, and expandable telemetry and data strategy is pivotal to the EAP. Machine learning (ML) and artificial intelligence (AI) strategies can then be leveraged to make the best decision on reserving new resources or optimizing the use of existing ones.

OCEAN ANALOGY—UNDERLAY NETWORKS WANT TO BE LIKE THE BOTTOM OF THE OCEAN—QUIET, SERENE, AND ALMOST UNCHANGING OVER TIME; BUT NEVERTHELESS A DARK AND HIGH-PRESSURE ENVIRONMENT.

AN UNDERLAY-NETWORK THAT TRIES TO REACT TO EACH REQUIREMENT OF EVERY APPLICATION ONLY BRINGS TURBULENT TORRENTS AND VORTICES FROM THE SURFACE DOWN TO THE SEABED.

SOFTWARE DEFINED APPLICATION CONNECTIVITY

While connectivity and reachability are assumed to be solved problems, many network teams still struggle with rationalizing their network fabric with the dynamic needs of the application layer. A platform must address these challenges as well.

A Case to Separate the Application Connectivity Plane

The issue with existing approaches is we translate application connectivity needs to enterprise WAN connectivity, especially in a distributed cloud. Approaches to addressing a distributed cloud network could use different SDN strategies or pure routing-based methods. But these solutions heavily rely on the homogeneity of the infrastructure and hence fall short on delivering a consistent behavior.

The only means by which we can achieve a globally scalable, secure, and stable network for the application is to separate the application connectivity plane from the underlying network, as discussed next.

Proposed Connectivity Stack

In the evolution towards a distributed cloud architecture, the emerging SDN pattern is to jointly orchestrate both the underlay and the overlay networks and enable end-to-end programmability of the application path.

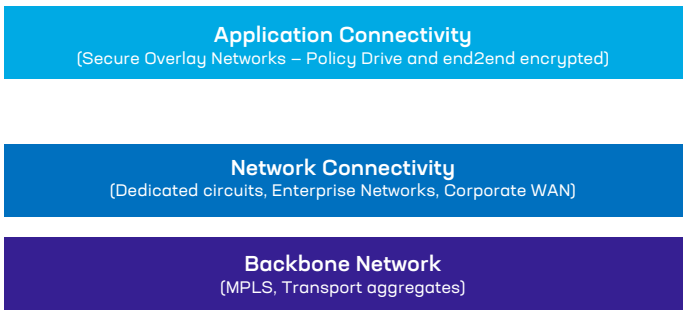

Figure 11: Recognizes that the application connectivity plane is different from the underlay (enterprise and backbone networks)

With Edge 2.0 we need to bring this semblance of stability even with connecting enterprise networks. We propose that the application connectivity plane must be separated from the enterprise network connectivity. The application connectivity pattern may or may not follow the same SDN connectivity as seen with data center overlay (for example, VXLAN, NVGRE, or others), or BGP-based SDN patterns.

Not all networks have been separated into overlay and underlay. Network teams are now seeing the need for this joint programmability as an important requirement to enable end-to-end automation, for which separation of the underlay and the overlay network is critical.

Separating the application plane from the underlay network and transport negates the need for the network teams to actively respond to every application request. Its scope becomes limited to providing robust, stable, and well-defined paths with a focus on optimizing the use of aggregate bandwidth at the lowest latencies.

ADAPTIVE INTERFACE MODEL

The key tenet of an EAP is that it is independent, autonomous, and manages a subset of the overall solution space. But EAPs need a means by which to communicate and offer their resources or request resources from other EAPs. There are several advantages of this approach.

Simplified Interface

An interface based on outcomes negates the need for the EAP to know the details about the other infrastructure. Assume there are two EAPs: A and B. Each EAP has a local scope of its infrastructure to schedule and reserve resources. EAP-A does not need to know

the resources provided by EAP-B. If, for example, EAP-A cannot satisfy the needs of the application and requires resources in the infrastructure which are in EAP-B's scope, then it can simply express its desired objective to EAP-B. It then becomes EAP-B's responsibility to invoke declarative or imperative interfaces to reserve, allocate, and configure from its free resources pool. EAP-B is also responsible to ensure the SLOs for that service within its infrastructure are met.

While a common syntax will be helpful to bootstrap an initial set of adaptive APIs, the implementation becomes more sophisticated and mature over time with the use of natural language processing and AI to reduce the required quality of input.

Simplified Operations

Different operational patterns also become simple where only the desired state needs to be specified. Resiliency patterns, for example, can merely be based on an expected SLO. It becomes the responsibility of the EAP providing the service to ensure the resources within its scope are allocated to meet the service level objectives. The calling EAP does not need to care, but would probably need to monitor the service metrics to know whether they are being met or not.

EAP FEATURES

- **Multi-Tenant**: While the mesh of EAPs showing in Figure 9 are for a single deployment, multiple EAPs per cloud are possible. Another application could talk to a different EAP or a mesh of EAPs.
- **Designed to Scale**: Each mesh is a peer-to-peer (P2P) mesh and can horizontally scale. Peer EAPs can independently keep track of which EAPs have what resource type and how to connect to the resource. AI/ML will further enhance how each EAP schedules resources within its own scope or reaches out to other EAPs as needed.
- **Built-in Networking**: For applications to connect across the distributed cloud, the EAP must support built-in networking agnostic of the underlying infrastructure.

Putting It Together

Figure 12 visualizes the role of different layers when deploying an application on distributed cloud.

The highlights of this representation are:

- Each EAP has a localized scope, and all of their capabilities are not indicated in Figure 10. The EAP must support the resource discovery and scheduler function. EAPs schedule resources by interfacing with the appropriate infrastructure controller via adaptive APIs.

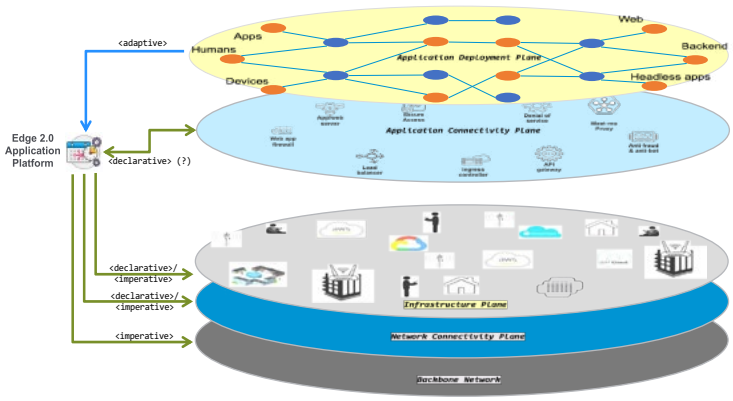

Figure 12: Reference showing role of different entities and layers.

- Application connectivity is not the same as network connectivity—like the service-mesh sidecar model, the resources to connect applications across multi-cloud must be part of the application infrastructure. One could argue the infrastructure plane also provides application network connectivity, thus, should be below the network layer. That would be technically right as well, but we chose to subsume it inside the “application connectivity” plane as these are primarily network functions.
- The infrastructure plane must be deemed separate from the application plane.
- Network connectivity is different from the transport, as we are essentially talking about specific routable networks which are separate from application connectivity.
- The transport plane can be thought of as aggregate backbone networks that can be provisioned, provided the telecom provider enables processes and APIs to connect the network layer above.

Summary

This white paper attempts to drill down a few more layers on the original Edge 2.0 manifesto. We have introduced some new themes, with the goal to inform different stakeholders within internal organizations and externally to the industry.

Edge 2.0 is built on the notion of a distributed cloud where every entity can participate as both the source, as well as the destination of services. The primary theme is that Edge 2.0 will be about the experience and not the asset or location.

The edge application platform (EAP) is introduced as a stack of capabilities to enable Edge 2.0 to be realized over a distributed cloud.

Implementation details have been deliberately skipped to present a vendor-agnostic view.

Resources

1 beyondcorp.com

2 https://www.f5.com/pdf/reports/f5-office-of-the-cto-report-continuous-api-sprawl.pdf

3 Wikipedia - An autonomous system (AS) is a collection of connected Internet Protocol (IP) routing prefixes under the control of one or more network operators on behalf of a single administrative entity or domain that presents a common, clearly defined routing policy to the Internet.