Data Cache & Edge Placement

Problem

As workloads move to different clouds and to the edge, accessing centralized data over WAN introduces unacceptable amounts of latency. At the same time, there are reservations to copying the data into the cloud (ingress/egress costs). This leaves most at an impasse—slowing cloud migration.

Solution

Deploy local data services into the edge node and leverage the low-latency, high-throughput, direct cloud connectivity (Network Step 3*) for access. This way the data can be accessed by applications from multiple clouds, partners, or from users coming in over the field area networks (however, it is not actually stored in the cloud). This private storage at the edge provides a different tier of performance than the data repository (Data Step 1). The type of implementation will depend on access volume, frequency and data size (in memory, solid state or hybrid) with asynchronization capabilities. Typical use cases involve a mix of data types—images (VMs, containers, application binaries and libraries), media content and workload data sets (e.g., batch data for a compute farm, etc.). Most providers have tools for centralized management (for this environment, a service API is also needed). This typically supports more block interfaces (FC, FCoE, iSCSI, NFS, pNFS, CIFS/SMB), and today an object/API.

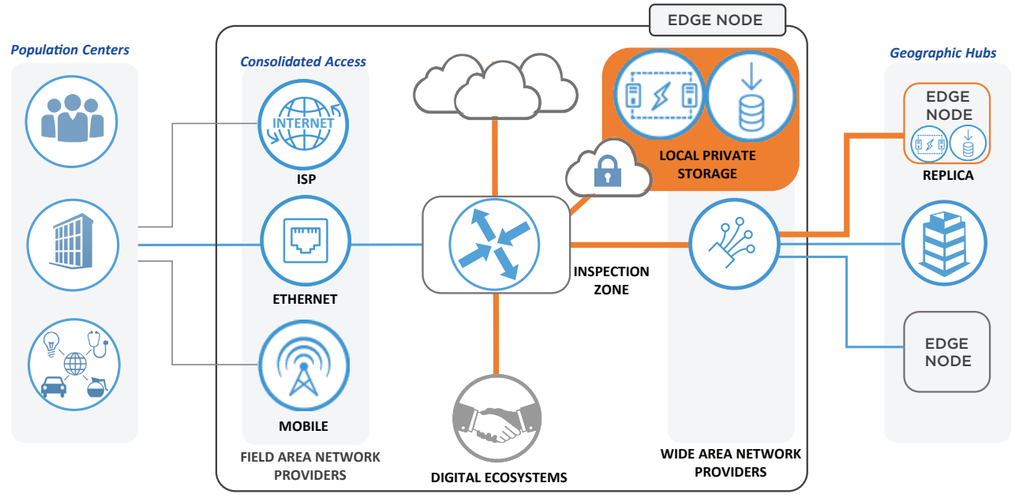
Constraints

1. Moving large datasets out to the edge can create management and accountability problems. Current infrastructure was not designed to support that and it can fall off the radar.
2. Workloads have been architected and sized based on local I/O expectations. Moving the workload without the data is rarely feasible.
3. The data is mostly unclassified. It might be able to go to cloud, or it might not. To move it may need CIO approval, which definitely slows down cloud migrations.
4. Traditional infrastructure had tightly coupled services, such that storage included backups, snapshots, off-site replicas, etc. Datasets in the cloud require the coverage.
5. Cost of storage is frequently misused in these discussions, which can prevent exploring solutions and alternatives.

Steps

1. Deploy the local private storage into the edge node. Add a second instance if you need failover/recovery. The second instance could also reside in a different edge node.
2. Replicate the nodes to each other.
3. Attach interfaces to the segmented networks they will be servicing (including cloud access).
4. Integrate with boundary control and the inspection zone (Security Blueprint*).
5. Register with the vendor(s) management tools and publish a self-service API.
6. Configure policies and integrate with policy management. Collect events and logging.
7. Configure daily snapshots.
8. To back it up, leverage the object interface to backup directly to the distributed repository (or direct to cloud storage (Step 1)).
9. Configure data to be pushed to it if being used as a cache.

Reference View



* Network and Security Blueprints — IOAKB.com

Forces

- Data needs to be moved closer to the edge, where business and customer engagement occurs.
- With changing regulations and increasing reports on data leakage, sensitivity levels to data breaches are high and the rules on certain data types can also change.
- Business is moving lots of functions to SaaS, which at times means having to encrypt and mask the data being stored.
- Storage concerns drive some renegade behavior (shadow IT) which, right or wrong, can expose the business to risk and contribute to data sprawl.

Results

- Localizing data removes the bulk of the latency and is best positioned at the intersection point.
- Running multicloud application workloads doesn't require moving data — access the data in the edge node over secure, low-latency connectivity.
- You have added layers of data protection so it scales without increasing risk.
- Can be used to migrate data between clouds.
- Act as a data exchange server for access to monetized data sets shared with partners across segmented networks.
- Additional streaming data and real-time data cache tools can use this service as a backing store, de-staging lower priority data.