BigQuery Omni started as a project to run Google BigQuery on AWS and Azure to expand the TAM for one of Google's most successful products. This 5+ year effort has begun to deliver differentiated customer innovation through cross-cloud joins and materialized views over data lakes after a significant upfront investment in multi-cloud infrastructure. This talk will cover the things we got right and wrong as an extension to our accepted SIGMOD'24 paper on BigQuery's Evolution toward a Multi-Cloud Lakehouse. These lessons are valuable for building and operating multi-cloud platforms as well as the evolution of legacy systems.

BigQuery is a legacy system by Google standards, having been first launched to the public in 2010. It is a hard multi-tenant cloud service depending on a distributed query engine (dremel), in-memory filesystem (MindMeld), RDMA (Snap), a control plane, and storage services (colossus) to name a few. These services receive continual improvements across the organization and receive 24x7 support from specialized SRE teams. Given the money maker lives on Borg, it became clear that we needed to focus on providing a Borg experience on AWS and Azure to keep up innovation.

In order to provide Borg to these services, we had to wrangle the compute (VMs), runtime environment (docker containers/core libraries), orchestration (Kubernetes), service discovery (chubby), authentication (ATLS), service connectivity (a flat network with Borg), and security/compliance infrastructure. A flat network is required to connect to monitoring/health services needed to onboard SRE teams (e.g. Monarch) and allows us to incrementally shift existing components from Borg to AWS/Azure as necessary to meet product needs.

Security proved to be the most significant focus area in our initial infrastructure build out. The operational requirement of the flat network and product requirement of expanding an existing service meant we were putting Google, BigQuery, and Omni at risk.

- We leveraged the overlay network used for YouTube CDN and built the infrastructure to grant our clusters their own namespaces to act as borg cells. This allowed us to take advantage of defense in depth for things like cross-region access and RPC-level security without creating a parallel track of security and networking tools.
- The Google Monorepo is the source of truth and Borg is the highest privilege actor in the system. All permissions on AWS are minimal with decisions like VM scaling/configuration being performed from Borg.
- Container security has not been sufficient defense in depth for our architecture but matches Borg. When our product requirements eventually shift to untrusted code we will need to invest in isolation such as gVisor, firecracker.
- Source transparency and insider risk is an increasingly critical issue in the industry, driving the team to port a Google-built Linux distribution, build our own Kubernetes distribution, develop auditing pipelines for the entirety of the system down from AWS events, network flow, SSH/exec actions on VMs, Kubernetes events, and application actions.

Developer velocity and cost efficiency were underinvested in the beginning of the project. The early developers quickly became experts in the system, had their workflows carved by late nights tracking down crash-looping binaries, and access to Sundar's credit card to pay bills. This does not scale as the team grew to deliver on a production product and were sent to work from home by a global pandemic.

- Opinionated developer tools for interacting with clusters significantly improved team velocity and hid cloud-specific knowledge that was not required for much of the team. Porting the developer stack on Google Cloud greatly reduced operating expenses.
- Diverging VM management and architecture across clouds drove significant cost and reliability improvements through AWS Auto Scaling Groups and AWS Graviton. Multi-cloud can not be successful if restricted to the lowest common denominator.